Presenting User Data with Table Views and Browsers

Session 110

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

Presenting User Data with Table Views and Browsers

Corbin Dunn
Cocoa Software Engineer

Introduction and Prerequisites

- An advanced talk focused on the AnimatedTableView sample code
- Expect you to have a firm knowledge of Cocoa development
  - Cocoa data sources and delegates
  - Familiar with custom cell creation/drawing
  - Knowledge of NSOutlineView/NSTableView/NSBrowser
- Sample code is for Snow Leopard
  - But most concepts work on Leopard

What We Are Covering

- Subclassing and enhancing NSTableView
  - Batch loading cell contents
  - Adding subviews
  - Custom cell editors
- Custom animations
  - Animating from a table view
  - Animating a pop-up window
- Custom NSBrowsers
  - Utilizing new Snow Leopard APIs

AnimatedTableView Sample Code

Use the sample code!

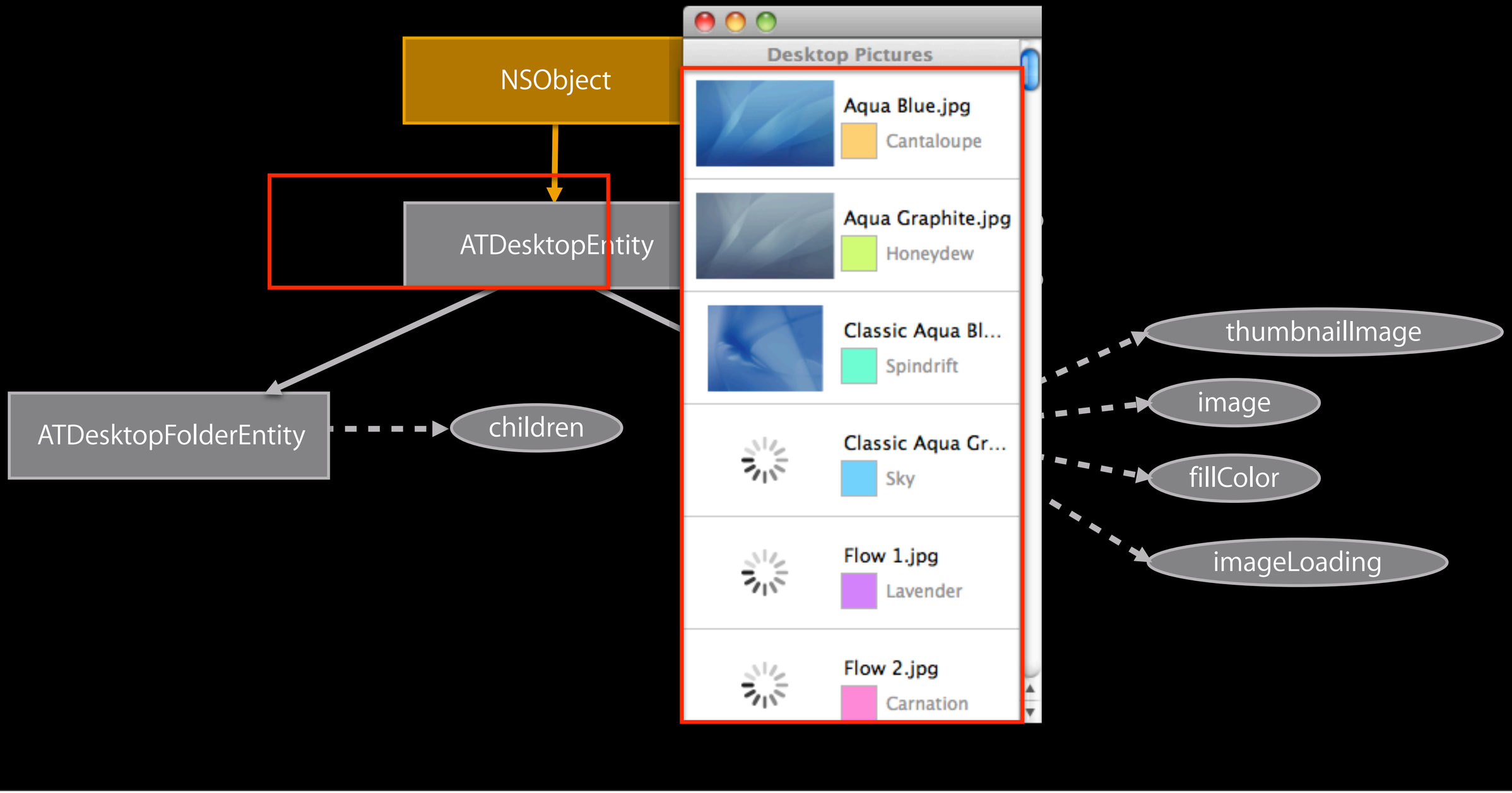
- “AnimatedTableView”—available at http://developer.apple.com
- Modify and reuse the code

Demo

AnimatedTableView sample code

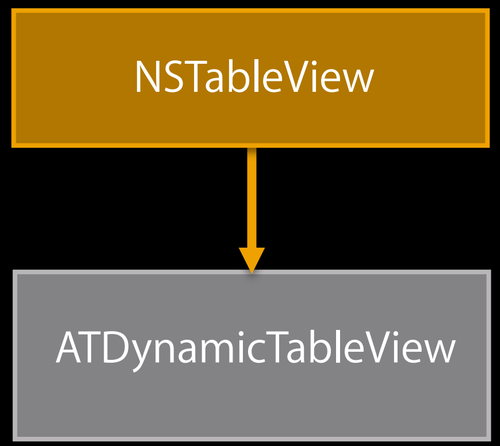
Primer—AnimatedTableView Architecture

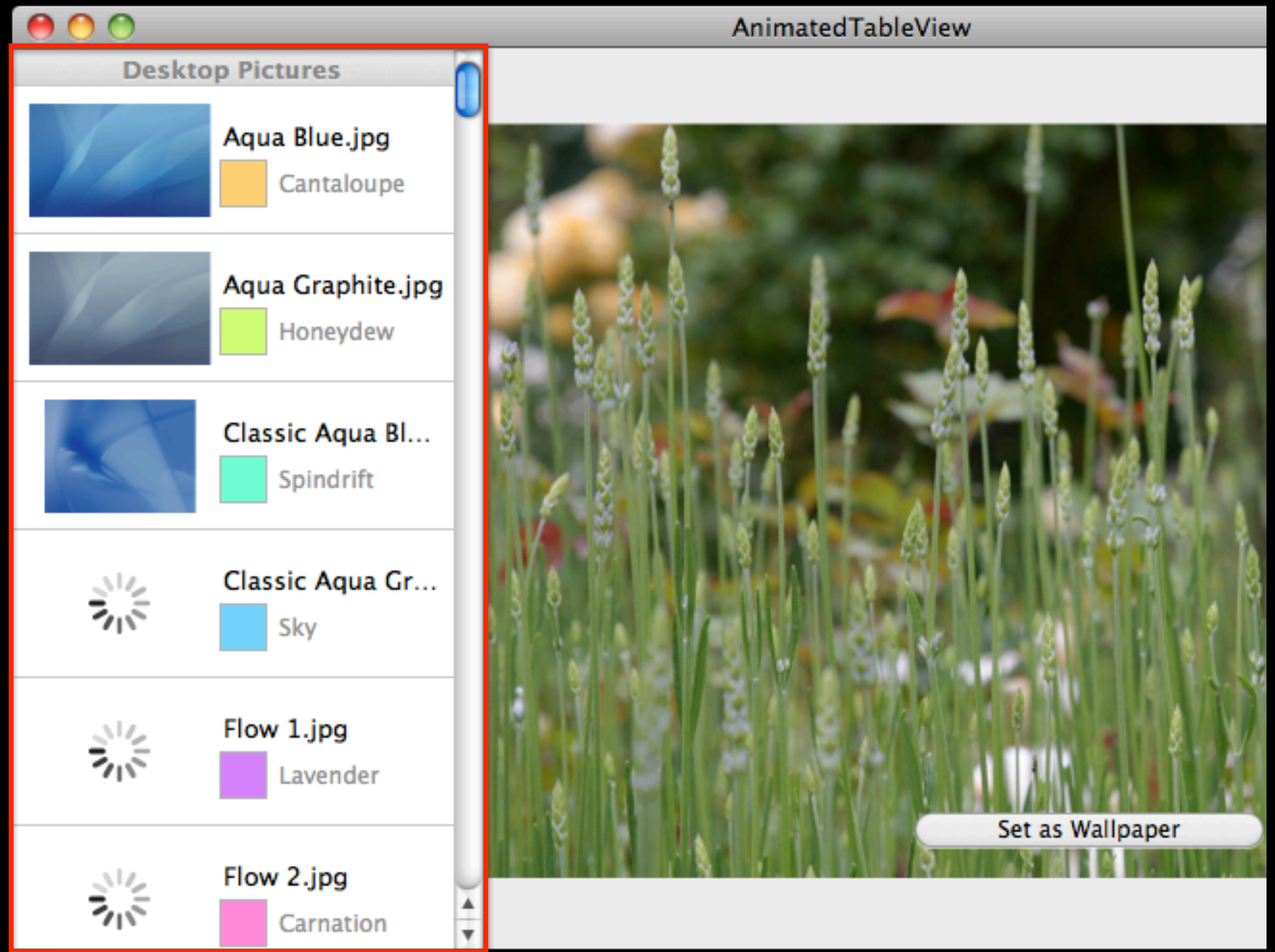
Model—Class hierarchy



Primer—AnimatedTableView Architecture

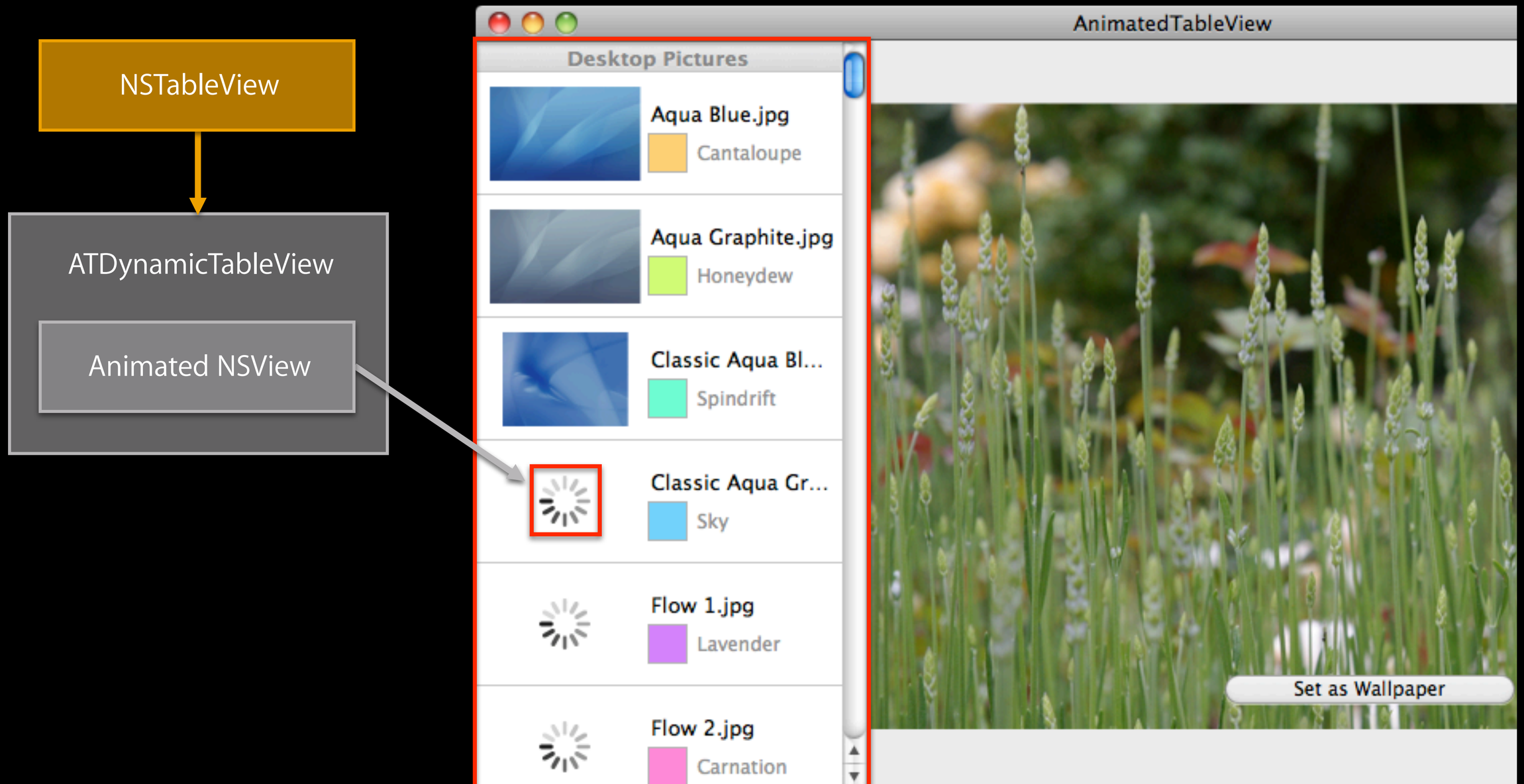
Views, cells, and windows





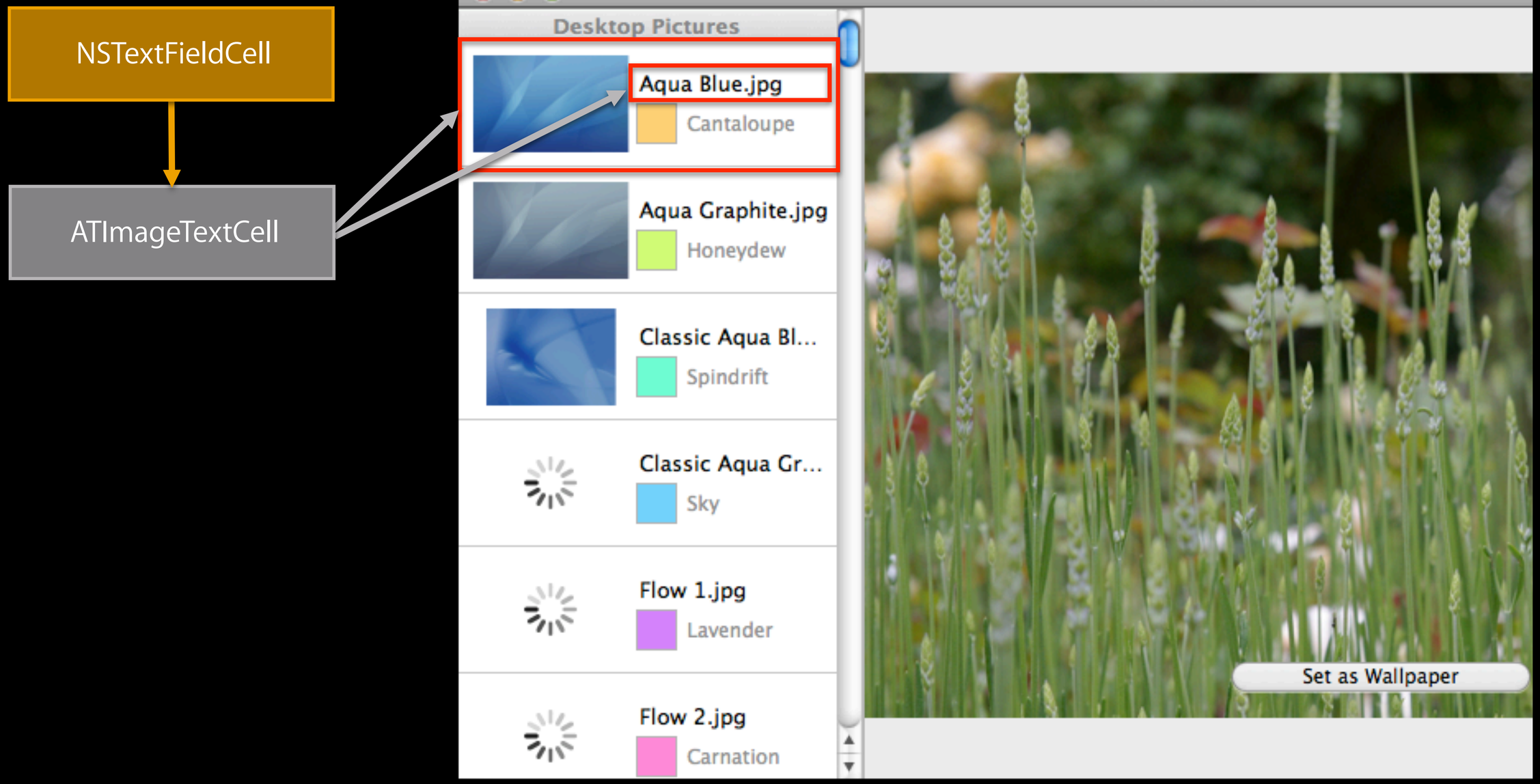
Primer—AnimatedTableView Architecture

Views, cells, and windows



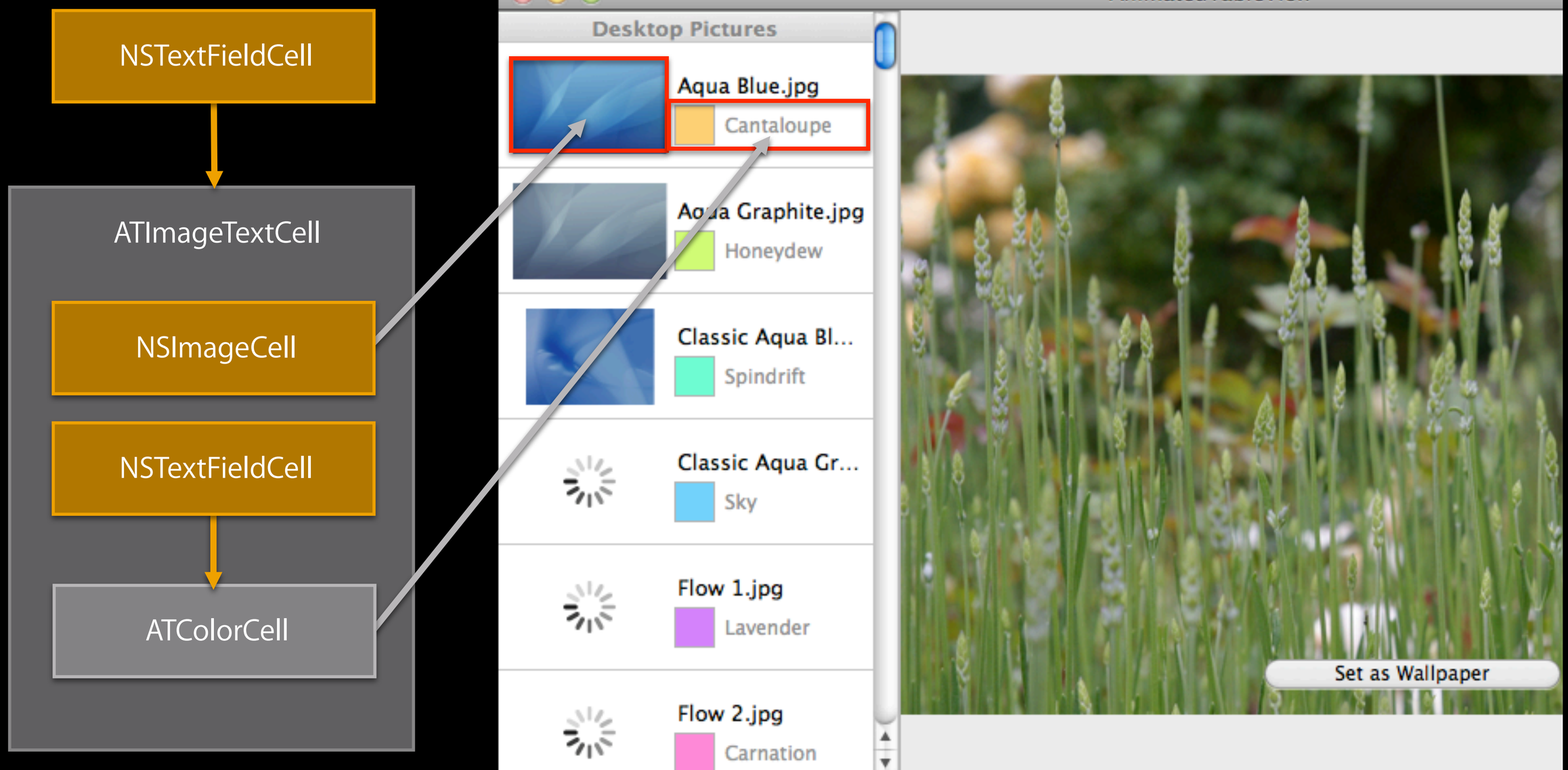
Primer—AnimatedTableView Architecture

Views, cells, and windows



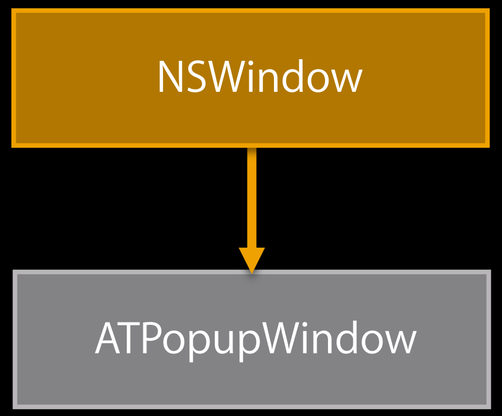
Primer—AnimatedTableView Architecture

Views, cells, and windows



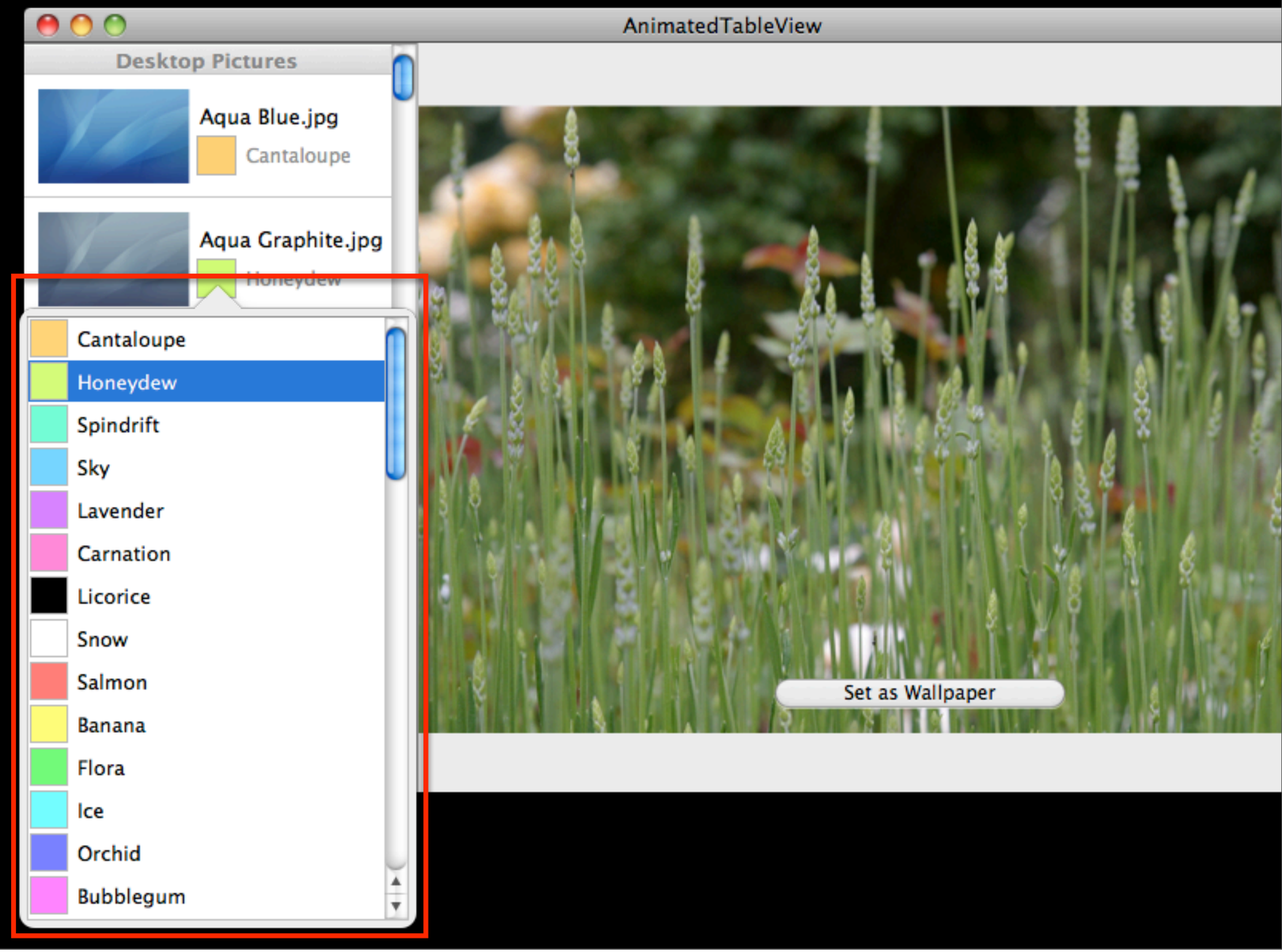
Primer—AnimatedTableView Architecture

Views, cells, and windows



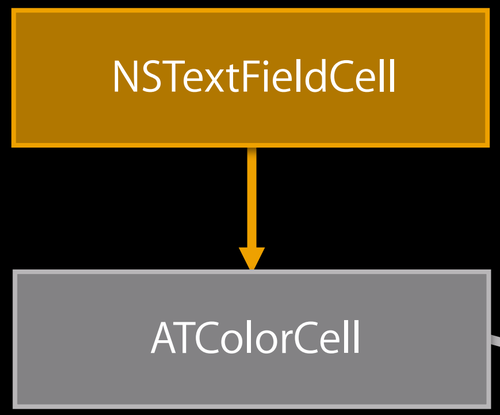
-canBecomeKeyWindow

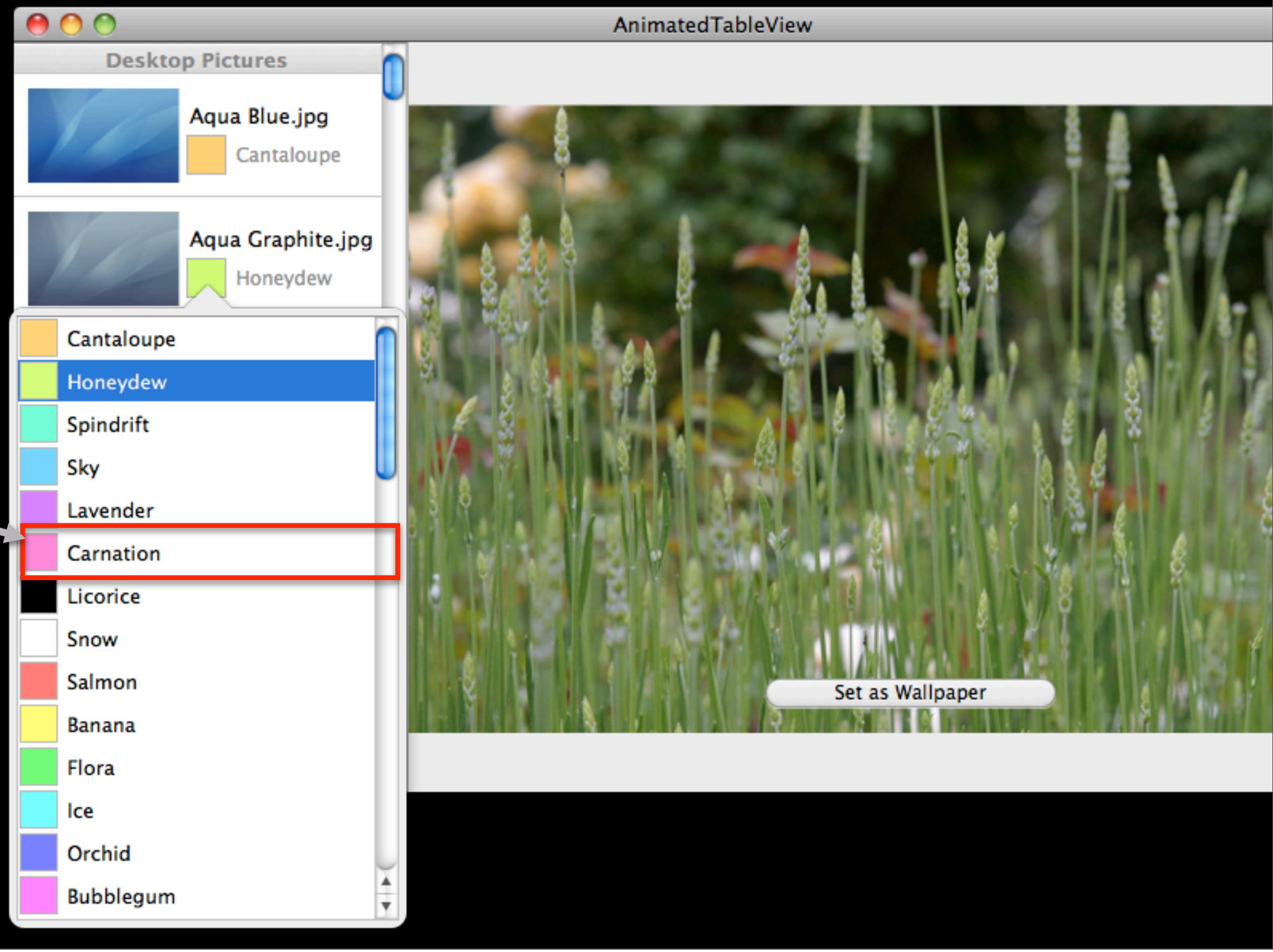
-popup animation



Primer—AnimatedTableView Architecture

Views, cells, and windows





Primer—AnimatedTableView Architecture

Controller—Hooks the view to the model

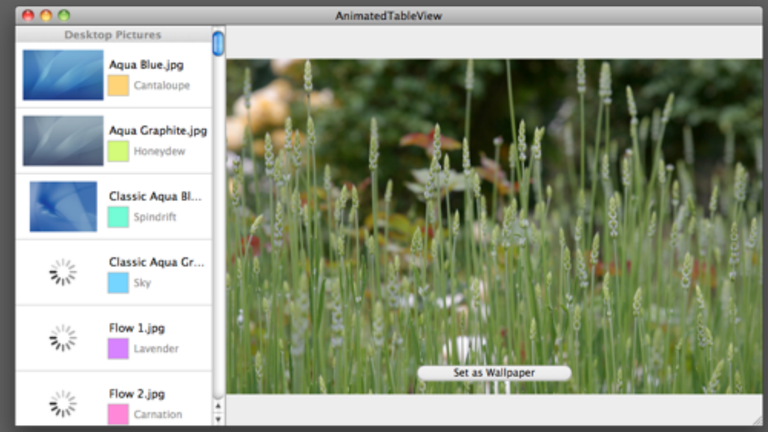
Model

ATDesktopFolderEntity

ATDesktopImageEntity

ATContentController

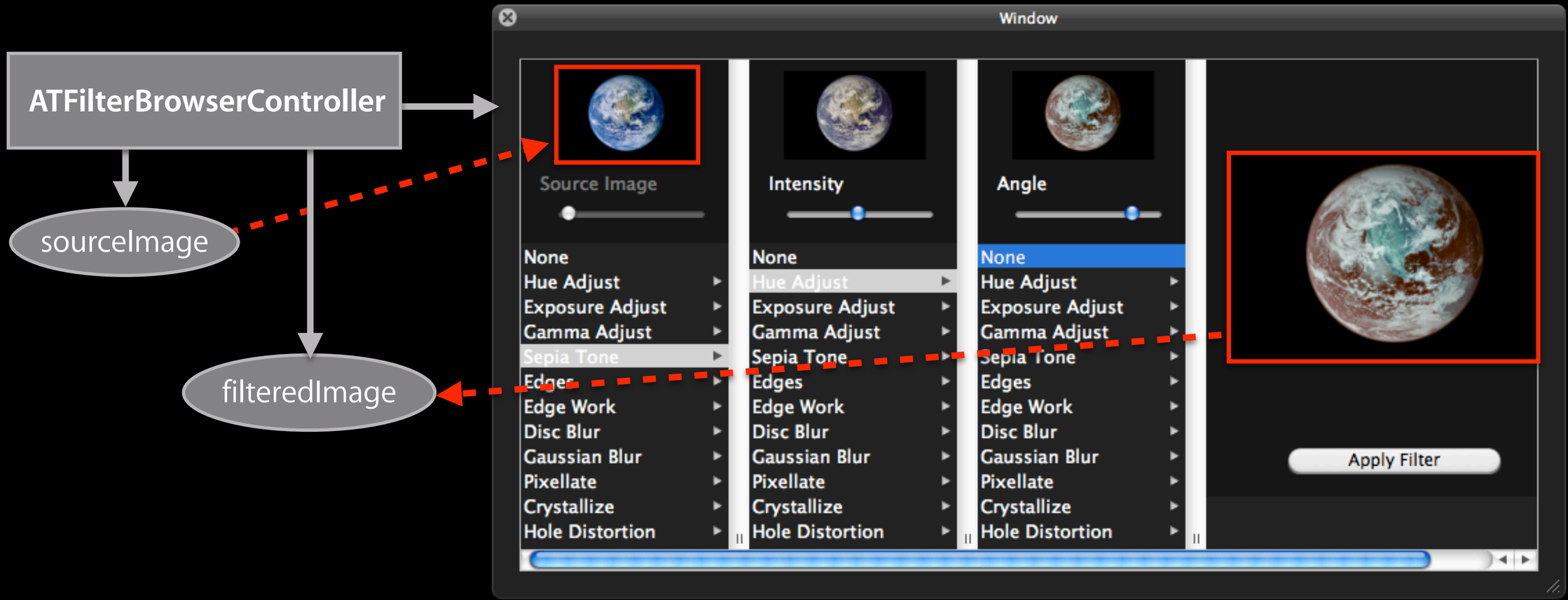
Views



Primer—AnimatedTableView Architecture

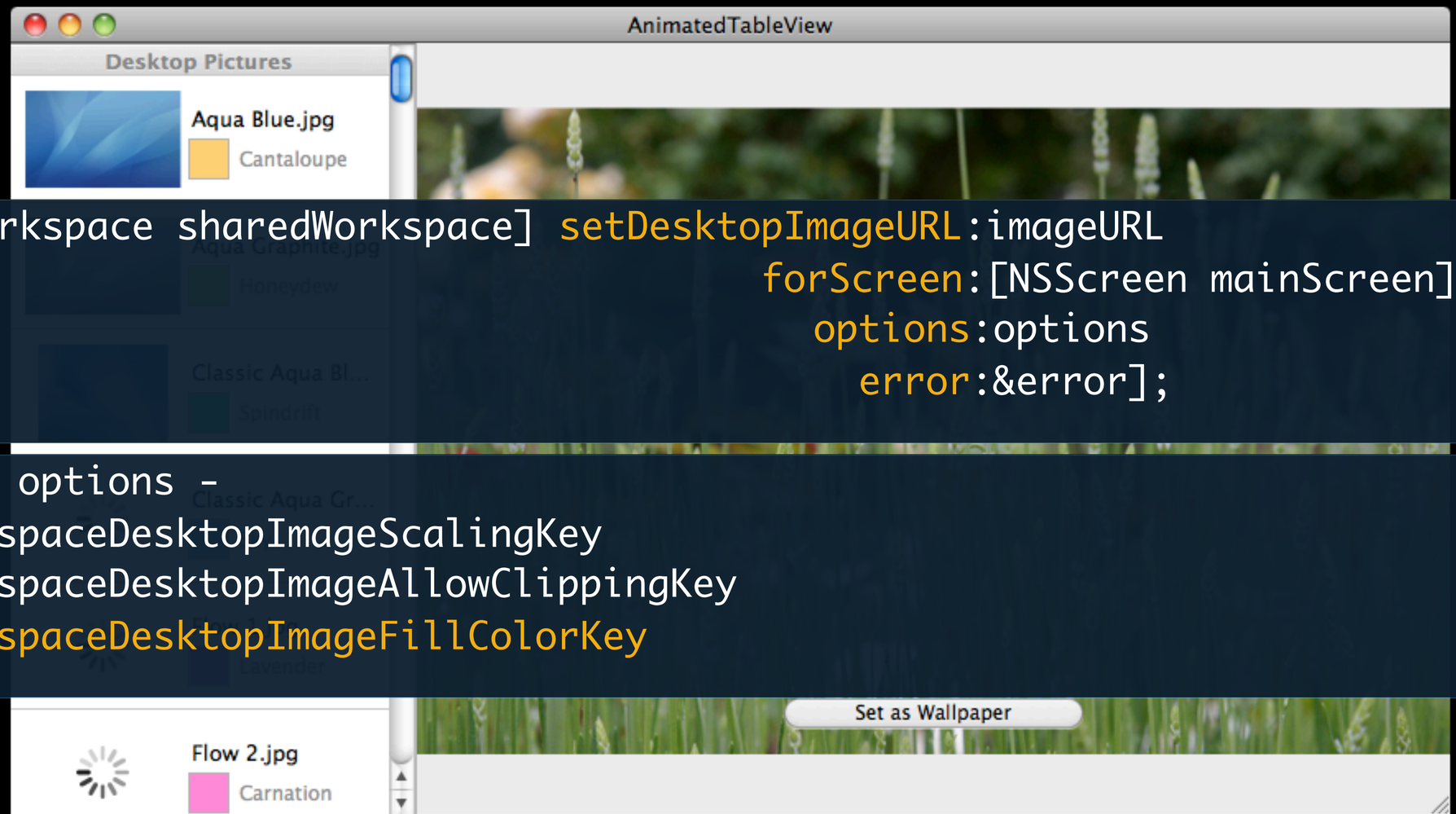
Views, cells, and windows

Another controller to show the image filter browser window



Primer—AnimatedTableView Architecture

Changing the wallpaper




```
[[NSWorkspace sharedWorkspace] setDesktopImageURL:imageURL
                                        forScreen:[NSScreen mainScreen]
                                          options:options
                                            error:&error];
```

```
Keys for options -
  NSWorkspaceDesktopImageScalingKey
  NSWorkspaceDesktopImageAllowClippingKey
  NSWorkspaceDesktopImageFillColorKey
```

Subclassing NSTableView

Batch loading cell contents

What We Are Covering

- Subclassing and enhancing NSTableView
  - Batch loading cell contents
  - Adding subviews
  - Custom cell editors
- Custom animations
  - Animating from a Table View
  - Animating a pop-up window
- Custom NSBrowsers
  - Utilizing new Snow Leopard APIs

Batch Loading Cell Contents

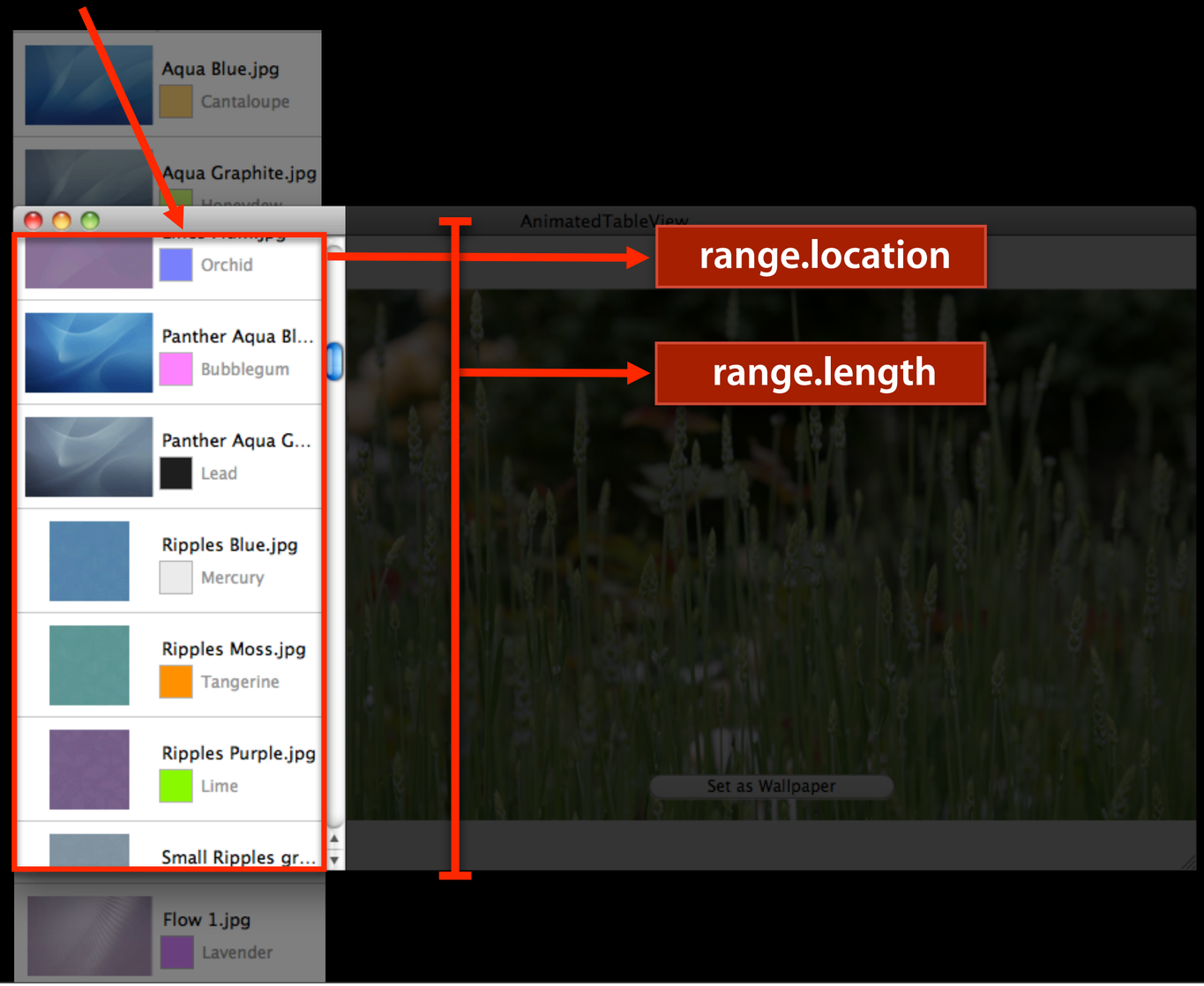
- **Don’t** rely on the delegate method for display only

```
- (void)tableView:(NSTableView *)tableView
  willDisplayCell:(id)cell
   forTableColumn:(NSTableColumn *)tableColumn
              row:(NSInteger)row;
```

  - It is called from -preparedCellAtColumn:row:
  - Used for more operations than drawing
    - Double-click to resize column
    - Type selection

Batch Loading Cell Contents

- Preload everything in the -visibleRect of an NSTableView
- Cache an NSRange for the visible rows



Batch Loading Cell Contents

- We will extend the NSTableView delegate with our own

```
@protocol ATDynamicTableViewDelegate <NSTableViewDelegate>
@optional

- (void)dynamicTableView:(ATDynamicTableView *)tableView
    changedVisibleRowsFromRange:(NSRange)oldVisibleRows
                        toRange:(NSRange)newVisibleRows;
@end
```

Batch Loading Cell Contents

- Subclass NSTableView and add ivar storage for the visible row range

```
@interface ATDynamicTableView : NSTableView {
@private
    NSRange _visibleRows;
}

@property(assign) id <ATDynamicTableViewDelegate> delegate;

@end
```

Batch Loading Cell Contents

- Override -viewWillDraw to inform the delegate as late as possible

```
- (void)viewWillDraw {
  [super viewWillDraw];
  NSRange newVisibleRows = [self rowsInRect:self.visibleRect];
  BOOL visibleRowsNeedsUpdate =
       !NSEqualRanges(newVisibleRows, _visibleRows);

  if (visibleRowsNeedsUpdate) {
      id delegate = [self delegate];
      [delegate dynamicTableView:self
         changedVisibleRowsFromRange:oldVisibleRows toRange:newVisibleRows];
      _visibleRows = newVisibleRows;
  }
}
```

Batch Loading Cell Contents

- Override -viewWillDraw to inform the delegate as late as possible

```
- (void)viewWillDraw {
  [super viewWillDraw];
  NSRange newVisibleRows = [self rowsInRect:self.visibleRect];
  BOOL visibleRowsNeedsUpdate =
       !NSEqualRanges(newVisibleRows, _visibleRows);

  if (visibleRowsNeedsUpdate) {
      id delegate = [self delegate];
      [delegate dynamicTableView:self
         changedVisibleRowsFromRange:oldVisibleRows toRange:newVisibleRows];
      _visibleRows = newVisibleRows;
  }
}
```

Batch Loading Cell Contents

- Override -viewWillDraw to inform the delegate as late as possible

```
- (void)viewWillDraw {
   [super viewWillDraw];
   NSRange newVisibleRows = [self rowsInRect:self.visibleRect];
   BOOL visibleRowsNeedsUpdate =
        !NSEqualRanges(newVisibleRows, _visibleRows);

   if (visibleRowsNeedsUpdate) {
       id delegate = [self delegate];
       [delegate dynamicTableView:self
          changedVisibleRowsFromRange:oldVisibleRows toRange:newVisibleRows];
       _visibleRows = newVisibleRows;
   }
}
```

Batch Loading Cell Contents

- Override -viewWillDraw to inform the delegate as late as possible

```
- (void)viewWillDraw {
  [super viewWillDraw];
  NSRange newVisibleRows = [self rowsInRect:self.visibleRect];
  BOOL visibleRowsNeedsUpdate =
       !NSEqualRanges(newVisibleRows, _visibleRows);

  if (visibleRowsNeedsUpdate) {
      id delegate = [self delegate];
      [delegate dynamicTableView:self
         changedVisibleRowsFromRange:oldVisibleRows toRange:newVisibleRows];
      _visibleRows = newVisibleRows;
  }
}
```

Batch Loading Cell Contents

- The delegate can batch load and use KVO to watch for changes of just the visible images

```
- (void)dynamicTableView:(ATDynamicTableView *)tableView
    changedVisibleRowsFromRange:(NSRange)oldVisibleRows
                        toRange:(NSRange)newVisibleRows {

    _observedVisibleItems = [[_tableContents
                                subarrayWithRange:newVisibleRows] retain];
    for (ATDesktopImageEntity *imageEntity in _observedVisibleItems) {
      [imageEntity loadImage];
      [imageEntity addObserver:self
                    forKeyPath:@"thumbnailImage"
                       options:0
                       context:NULL];
    }
}
```

Batch Loading Cell Contents

- The delegate can batch load and use KVO to watch for changes of just the visible images

```
- (void)dynamicTableView:(ATDynamicTableView *)tableView
    changedVisibleRowsFromRange:(NSRange)oldVisibleRows
                        toRange:(NSRange)newVisibleRows {

    _observedVisibleItems = [[_tableContents
                                subarrayWithRange:newVisibleRows] retain];
    for (ATDesktopImageEntity *imageEntity in _observedVisibleItems) {
      [imageEntity loadImage];
      [imageEntity addObserver:self
                    forKeyPath:@"thumbnailImage"
                       options:0
                       context:NULL];
    }
}
```

Batch Loading Cell Contents

- The delegate can batch load and use KVO to watch for changes of just the visible images

```
- (void)dynamicTableView:(ATDynamicTableView *)tableView
    changedVisibleRowsFromRange:(NSRange)oldVisibleRows
                        toRange:(NSRange)newVisibleRows {

    _observedVisibleItems = [[_tableContents
                                subarrayWithRange:newVisibleRows] retain];
    for (ATDesktopImageEntity *imageEntity in _observedVisibleItems) {
      [imageEntity loadImage];
      [imageEntity addObserver:self
                    forKeyPath:@"thumbnailImage"
                       options:0
                       context:NULL];
    }
}
```

Batch Loading Cell Contents

- The delegate can batch load and use KVO to watch for changes of just the visible images

```
- (void)dynamicTableView:(ATDynamicTableView *)tableView
    changedVisibleRowsFromRange:(NSRange)oldVisibleRows
                        toRange:(NSRange)newVisibleRows {

    // First, stop observing all prior things
    for (ATDesktopEntity *imageEntity in _observedVisibleItems) {
        [imageEntity removeObserver:self forKeyPath:@"thumbnailImage"];
    }

    [_observedVisibleItems release];
    _observedVisibleItems = ...
```

Model Asynchronously Loads Images

The model does the heavy lifting with NSOperationQueue:

```
- (void)loadImage {
  @synchronized (self) {
    if (self.image == nil && !self.imageLoading) {
      self.imageLoading = YES;
      [ATSharedOperationQueue addOperationWithBlock:^(void) {
        NSImage *image = [[NSImage alloc] initWithContentsOfURL:self.fileURL];
      if (image != nil) {
        NSImage *thumbnailImage = ATThumbnailImageFromImage(image);
        // We synchronize access to the image/imageLoading pair of variables
        @synchronized (self) {
           self.imageLoading = NO;
           self.image = image;
           self.thumbnailImage = thumbnailImage;
        }
        [image release];
    }
  }];
```

Model Asynchronously Loads Images

The model does the heavy lifting with NSOperationQueue:

```
- (void)loadImage {
  @synchronized (self) {
    if (se                                    ageLoading) {
      self.imageLoadin = YES;
      [ATSharedOperationQueue addOperationWithBlock:^(void) {
        NSImage *image = [[NSImage alloc] initWithContentsOfURL:self.fileURL];
        if (image != nil) {
          NSImage *thumbnailImage = ATThumbnailImageFromImage(image);
          // We synchronize access to the image/imageLoading pair of variables
          @synchronized (self) {
             self.imageLoading = NO;
             self.image = image;
             self.thumbnailImage = thumbnailImage;
          }
          [image release];
        }
      }];
```

Shared global NSOperationQueue

Model Asynchronously Loads Images

The model does the heavy lifting with NSOperationQueue:

```
- (void)loadImage {
  @synchronized (self) {
    if (self.image == nil && !self.imageLoading) {
      self.imageLoading = YES;
      [ATSharedOpera                                    (void) {
        NSImage *image = [[NSImage al oc] initWithContentsOfURL:self.fileURL];
        if (image != nil) {
          NSImage *thumbnailImage = ATThumbnailImageFromImage(image);
          // We synchronize access to the image/imageLoading pair of variables
          @synchronized (self) {
              self.imageLoading = NO;
              self.image = image;
              self.thumbnailImage = thumbnailImage;
          }
          [image release];
        }
      }];
```

Potentially blocking and slow operation

Model Asynchronously Loads Images

The model does the heavy lifting with NSOperationQueue:

```
- (void)loadImage {
  @synchronized (self) {
    if (self.image == nil && !self.imageLoading) {
      self.imageLoading = YES;
      [ATSharedOperationQueue addOperationWithBlock:^(void) {
        NSImage *image = [[NSImage alloc] initWithContentsOfURL:self.fileURL];
        if (image != nil) {
          NSImage *thumbnailImage = ATThumbnailImageFromImage(image);
          // We synchronize access to the image/imageLoading pair of variables
          @synchronized (self) {
             self.imageLoading = NO;
             self.image = image;
             self.thumbnailImage = thumbnailImage;
          }
          [image release];
        }
      }];
```

Model Asynchronously Loads Images

The model does the heavy lifting with NSOperationQueue:

```
- (void)loadImage {
  @synchronized (self) {
    if (self.image == nil && !self.imageLoading) {
      self.imageLoading = YES;
      [ATSharedOperationQueue addOperationWithBlock:^(void) {
        NSImage *image = [[NSImage alloc] initWithContentsOfURL:self.fileURL];
        if (image != nil) {
          NSImage *thumbnailImage = ATThumbnailImageFromImage(image);
          // We synchronize access to the image/imageLoading pair of variables
          @synchronized (self) {
              self.imageLoading = NO;
              self.image = image;
              self.thumbnailImage = thumbnailImage;
          }
          [image release];
        }
      }];
```

Model Asynchronously Loads Images

The model does the heavy lifting with NSOperationQueue:

```
- (void)loadImage {
  @synchronized (self) {
    if (self.image == nil && !self.imageLoading) {
      self.imageLoading = YES;
      [ATSharedOperationQueue addOperationWithBlock:^(void) {
        NSImage *image = [[NSImage alloc] initWithContentsOfURL:self.fileURL];
        if (image != nil) {
          NSImage *thumbnailImage = ATThumbnailImageFromImage(image);
          // We synchronize access to the image/imageLoading pair of variables
          @syn
              self.imageLoading = NO;
              self.image = image;
              self.thumbnailImage = thumbnailImage;
          }
          [image release];
      }
    }];
```

Sends off KVO notification

Controller Reloads via KVO Notification

Ensure we change the UI only on the main thread

```
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object change:(NSDictionary *)change
                       context:(void *)context {
    [self performSelectorOnMainThread:@selector(_reloadRowForEntity:)
                           withObject:object
                        waitUntilDone:NO
                                modes:[NSArray arrayWithObject:NSRunLoopCommonModes]];
}

- (void)_reloadRowForEntity:(id)object {
    NSInteger row = [_tableContents indexOfObject:object];
    NSIndexSet *rowIndexSet = [NSIndexSet indexSetWithIndex:row];
    [_tableViewMain reloadDataForRowIndexes:rowIndexSet
                              columnIndexes:[NSIndexSet indexSetWithIndex:0]];
}
```

Controller Reloads via KVO Notification

Ensure we change the UI only on the main thread

```
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object change:(NSDictionary *)change
                       context:(void *)context {
    [self performSelectorOnMainThread:@selector(_reloadRowForEntity:)
                           withObject:object
                        waitUntilDone:NO
                                modes:[NSArray arrayWithObject:NSRunLoopCommonModes]];
}

- (void)_reloadRowForEntity:(id)object {
    NSInteger row = [_tableContents indexOfObject:object];
    NSIndexSet *rowIndexSet = [NSIndexSet indexSetWithIndex:row];
    [_tableViewMain reloadDataForRowIndexes:rowIndexSet
                              columnIndexes:[NSIndexSet indexSetWithIndex:0]];
}
```

Controller Reloads via KVO Notification

Ensure we change the UI only on the main thread

```
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object change:(NSDictionary *)change
                       context:(void *)context {
    [self performSelectorOnMainThread:@selector(_reloadRowForEntity:)
                           withObject:object
                        waitUntilDone:NO
                                modes:[NSArray arrayWithObject:NSRunLoopCommonModes]];
}

- (void)_reloadRowForEntity:(id)object {
    NSInteger row = [_tableContents indexOfObject:object];
    NSIndexSet *rowIndexSet = [NSIndexSet indexSetWithIndex:row];
    [_tableViewMain reloadDataForRowIndexes:rowIndexSet
                              columnIndexes:[NSIndexSet indexSetWithIndex:0]];
}
```

Controller Reloads via KVO Notification

Ensure we change the UI only on the main thread

```
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object change:(NSDictionary *)change
                       context:(void *)context {
    [self performSelectorOnMainThread:@selector(_reloadRowForEntity:)
                           withObject:object
                        waitUntilDone:NO
                                modes:[NSArray arrayWithObject:NSRunLoopCommonModes]];
}

- (void)_reloadRowForEntity:(id)object {
    NSInteger row = [_tableContents indexOfObject:object];
    NSIndexSet *rowIndexSet = [NSIndexSet indexSetWithIndex:row];
    [_tableViewMain reloadDataForRowIndexes:rowIndexSet
                                 columnIndexes:[NSIndexSet indexSetWithIndex:0]];
}
```

Controller Reloads via KVO Notification

Ensure we change the UI only on the main thread

```
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object change:(NSDictionary *)change
                       context:(void *)context {
    [self performSelectorOnMainThread:@selector(_reloadRowForEntity:)
                           withObject:object
                        waitUntilDone:NO
                                modes:[NSArray arrayWithObject:NSRunLoopCommonModes]];
}

- (void)_reloadRowForEntity:(id)object {
    NSInteger row = [_tableContents indexOfObject:object];
    NSIndexSet *rowIndexSet = [NSIndexSet indexSetWithIndex:row];
    [_tableViewMain reloadDataForRowIndexes:rowIndexSet
                              columnIndexes:[NSIndexSet indexSetWithIndex:0]];
}
```

Subclassing NSTableView

Adding subviews to the cells

What We Are Covering

- Subclassing and enhancing NSTableView
  - Batch loading cell contents
  - Adding subviews
  - Custom cell editors
- Custom animations
  - Animating from a table view
  - Animating a pop-up window
- Custom NSBrowsers
  - Utilizing new Snow Leopard APIs

Demo

Subviews in a table view

Adding Subviews to NSTableView Cells

- Problems
  - No easy way to add an animation to a table view
  - Examples: NSProgressIndicator, NSPathControl
- Goals
  - Easy-to-use delegate API for adding subviews
  - Only views in the visible rect are added for performance
  - NSTableView subclass manages the views

Adding Subviews to NSTableView Cells

- Another optional delegate method

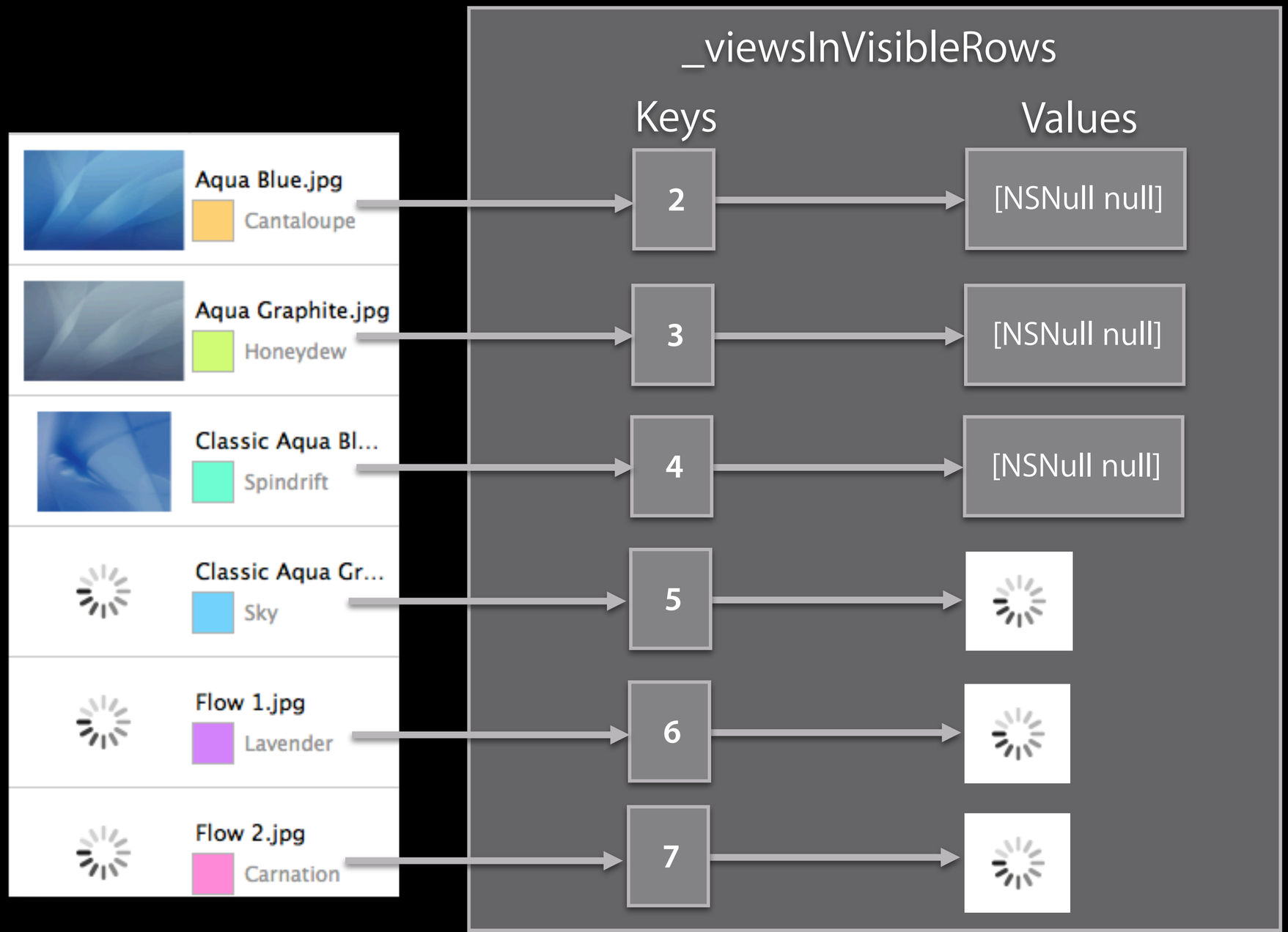
```
@protocol ATDynamicTableViewDelegate <NSTableViewDelegate>
@optional

...

- (NSView *)dynamicTableView:(ATDynamicTableView *)tableView
                 viewForRow:(NSInteger)row;

@end
```

Adding Subviews to NSTableView Cells

- The ATDynamicTableView subclass will keep track of views in a dictionary

```
@interface ATDynamicTableView : NSTableView {
@private
    NSRange _visibleRows;
    NSMutableDictionary *_viewsInVisibleRows;
    BOOL _viewsNeedUpdate;
}

@property(assign) id <ATDynamicTableViewDelegate> delegate;

@end
```

Adding Subviews to NSTableView Cells



Adding Subviews to NSTableView Cells

Use -viewWillDraw to do lazy subview addition

```
- (void)viewWillDraw {
  [super viewWillDraw];
  // <snip> Code shown earlier to update _visibleRows
  // _viewsNeedUpdate is also set to YES if the _visibleRows changed
  if (_viewsNeedUpdate) {
    _viewsNeedUpdate = NO;

    // First, remove any views that are no longer visible
    NSMutableIndexSet *rowIndexesToRemove =
              [NSMutableIndexSet indexSetWithIndexesInRange:oldVisibleRows];
    // Remove any rows from the set that are STILL visible
    [rowIndexesToRemove removeIndexesInRange:newVisibleRows];

    // Iterate and remove those views
    for (NSInteger row = [rowIndexes firstIndex];
                   row != NSNotFound;
                   row = [rowIndexes indexGreaterThanIndex:row]) {
```

Adding Subviews to NSTableView Cells

New Snow Leopard API

- Other key places to update and remove views

```
-reloadData
-noteHeightOfRowsWithIndexesChanged:(NSIndexSet *)indexSet
-reloadDataForRowIndexes:(NSIndexSet *)rowIndexes
            columnIndexes:(NSIndexSet *)columnIndexes
```

Delegate Creates the View

```
- (NSView *)dynamicTableView:(ATDynamicTableView *)tableView
                 viewForRow:(NSInteger)row {
  // Return a spinner for rows that are loading
  NSProgressIndicator *result = nil;
  ATDesktopImageEntity *entity = [self _imageEntityForRow:row];
  if (entity.thumbnailImage == nil) {
    result = [[[NSProgressIndicator alloc] initWithFrame:...] autorelease];
    // SNIP: Setting up the properties on the progress indicator
    NSRect progressFrame = [result frame];
    // Position it based on the cell frame location
    NSRect cellFrame = [tableView frameOfCellAtColumn:0 row:row];
    progressFrame.origin.x = NSMinX(cellFrame) +
            floor((NSWidth(cellFrame) - NSWidth(progressFrame)) / 2.0);
    progressFrame.origin.y = NSMinY(cellFrame) +
            floor((NSHeight(cellFrame) - NSHeight(progressFrame)) / 2.0);
    [result setFrame:progressFrame];
  }
  return result;
}
```

Delegate Creates the View

```
- (NSView *)dynamicTableView:(ATDynamicTableView *)tableView
                 viewForRow:(NSInteger)row {
  // Return a spinner for rows that are loading
  NSProgressIndicator *result = nil;
  ATDesktopImageEntity *entity = [self _imageEntityForRow:row];
  if (entity.thumbnailImage == nil) {
    result = [[[NSProgressIndicator alloc] initWithFrame:...] autorelease];
    // SNIP: Setting up the properties on the progress indicator
    NSRect progressFrame = [result frame];
    // Position it based on the cell frame location
    NSRect cellFrame = [tableView frameOfCellAtColumn:0 row:row];
    progressFrame.origin.x = NSMinX(cellFrame) +
            floor((NSWidth(cellFrame) - NSWidth(progressFrame)) / 2.0);
    progressFrame.origin.y = NSMinY(cellFrame) +
            floor((NSHeight(cellFrame) - NSHeight(progressFrame)) / 2.0);
    [result setFrame:progressFrame];
  }
  return result;
}
```

Delegate Creates the View

```
- (NSView *)dynamicTableView:(ATDynamicTableView *)tableView
                 viewForRow:(NSInteger)row {
  // Return a spinner for rows that are loading
  NSProgressIndicator *result = nil;
  ATDesktopImageEntity *entity = [self _imageEntityForRow:row];
  if (entity.thumbnailImage == nil) {
    result = [[[NSProgressIndicator alloc] initWithFrame:...] autorelease];
    // SNIP: Setting up the properties on the progress indicator
    NSRect progressFrame = [result frame];
    // Position it based on the cell frame location
    NSRect cellFrame = [tableView frameOfCellAtColumn:0 row:row];
    progressFrame.origin.x = NSMinX(cellFrame) +
            floor((NSWidth(cellFrame) - NSWidth(progressFrame)) / 2.0);
    progressFrame.origin.y = NSMinY(cellFrame) +
            floor((NSHeight(cellFrame) - NSHeight(progressFrame)) / 2.0);
    [result setFrame:progressFrame];
  }
  return result;
}
```

Delegate Creates the View

```
- (NSView *)dynamicTableView:(ATDynamicTableView *)tableView
                 viewForRow:(NSInteger)row {
  // Return a spinner for rows that are loading
  NSProgressIndicator *result = nil;
  ATDesktopImageEntity *entity = [self _imageEntityForRow:row];
  if (entity.thumbnailImage == nil) {
    result = [[[NSProgressIndicator alloc] initWithFrame:...] autorelease];
    // SNIP: Setting up the properties on the progress indicator
    NSRect progressFrame = [result frame];
    // Position it based on the cell frame location
    NSRect cellFrame = [tableView frameOfCellAtColumn:0 row:row];
    progressFrame.origin.x = NSMinX(cellFrame) +
            floor((NSWidth(cellFrame) - NSWidth(progressFrame)) / 2.0);
    progressFrame.origin.y = NSMinY(cellFrame) +
            floor((NSHeight(cellFrame) - NSHeight(progressFrame)) / 2.0);
    [result setFrame:progressFrame];
  }
  return result;
}
```

Delegate Creates the View

```
- (NSView *)dynamicTableView:(ATDynamicTableView *)tableView
                 viewForRow:(NSInteger)row {
  // Return a spinner for rows that are loading
  NSProgressIndicator *result = nil;
  ATDesktopImageEntity *entity = [self _imageEntityForRow:row];
  if (entity.thumbnailImage == nil) {
    result = [[[NSProgressIndicator alloc] initWithFrame:...] autorelease];
    // SNIP: Setting up the properties on the progress indicator
    NSRect progressFrame = [result frame];
    // Position it based on the cell frame location
    NSRect cellFrame = [tableView frameOfCellAtColumn:0 row:row];
    progressFrame.origin.x = NSMinX(cellFrame) +
            floor((NSWidth(cellFrame) - NSWidth(progressFrame)) / 2.0);
    progressFrame.origin.y = NSMinY(cellFrame) +
            floor((NSHeight(cellFrame) - NSHeight(progressFrame)) / 2.0);
    [result setFrame:progressFrame];
  }
  return result;
}
```

Delegate Creates the View

```
- (NSView *)dynamicTableView:(ATDynamicTableView *)tableView
                 viewForRow:(NSInteger)row {
  // Return a spinner for rows that are loading
  NSProgressIndicator *result = nil;
  ATDesktopImageEntity *entity = [self _imageEntityForRow:row];
  if (entity.thumbnailImage == nil) {
    result = [[[NSProgressIndicator alloc] initWithFrame:...] autorelease];
    // SNIP: Setting up the properties on the progress indicator
    NSRect progressFrame = [result frame];
    // Position it based on the cell frame location
    NSRect cellFrame = [tableView frameOfCellAtColumn:0 row:row];
    progressFrame.origin.x = NSMinX(cellFrame) +
            floor((NSWidth(cellFrame) - NSWidth(progressFrame)) / 2.0);
    progressFrame.origin.y = NSMinY(cellFrame) +
            floor((NSHeight(cellFrame) - NSHeight(progressFrame)) / 2.0);
    [result setFrame:progressFrame];
  }
  return result;
}
```

Subclassing NSTableView

Custom cell editors

What We Are Covering

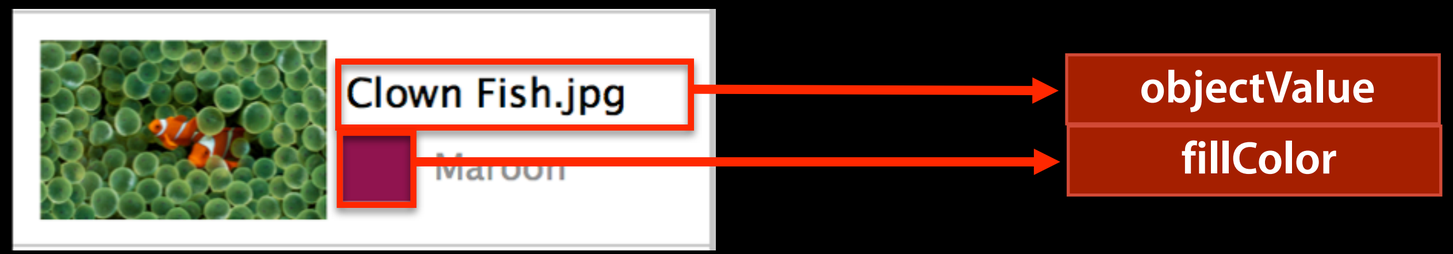
- Subclassing and enhancing NSTableView
  - Batch loading cell contents
  - Adding subviews
  - Custom cell editors
- Custom animations
  - Animating from a table view
  - Animating a pop-up window
- Custom NSBrowsers
  - Utilizing new Snow Leopard APIs

Demo

Custom cell editors

Problems to Solve

- Changing a single value in a multivalued cell



- Informing the delegate or data source of the change
- Editing a table view's cell outside of normal tracking

How to Solve the Problems

Cell talks to the view

- The table view subclass needs a method for the cell to call

```
@interface ATDynamicTableView : NSTableView { }

- (void)willStartEditingProperty:(NSString *)propertyName
                         forCell:(NSCell *)cell;

- (void)didEndEditingProperty:(NSString *)propertyName
                      forCell:(NSCell *)cell
                   successfully:(BOOL)success;

@end
```

How to Solve the Problems

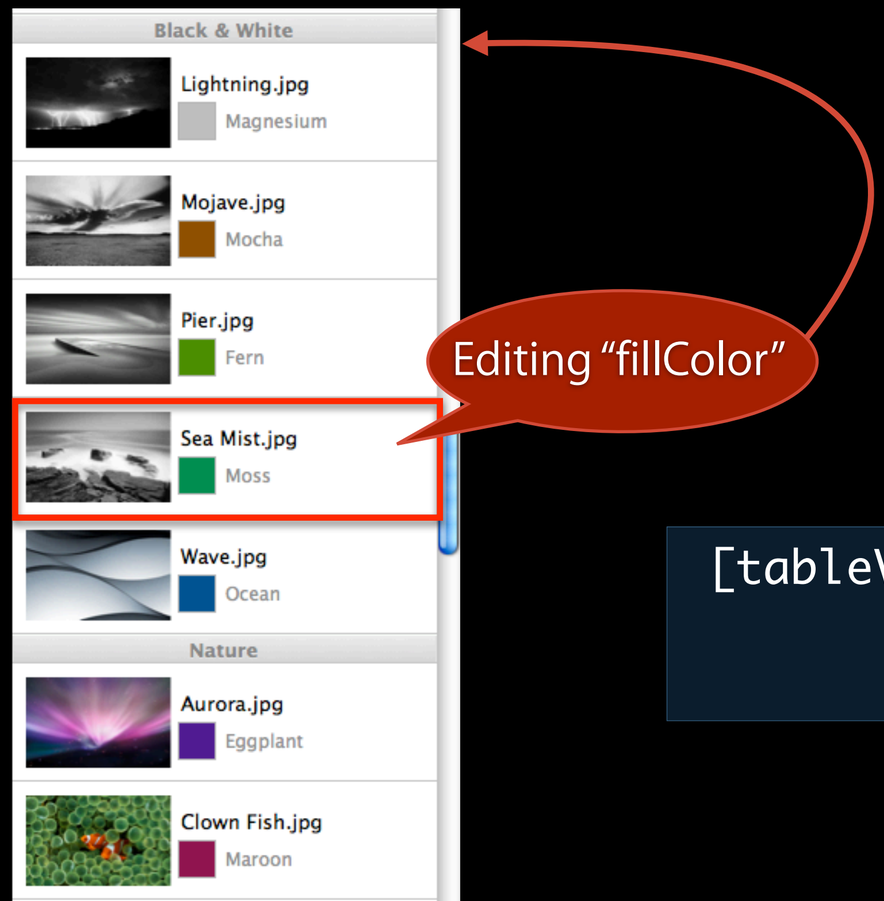
Cell talks to the view

- The table view subclass needs a method for the cell to call

```
@interface ATDynamicTableView : NSTableView { }

- (void)willStartEditingProperty:(NSString *)propertyName
                         forCell:(NSCell *)cell;

- (void)didEndEditingProperty:(NSString *)propertyName
                      forCell:(NSCell *)cell
                   successfully:(BOOL)success;

@end
```

How to Solve the Problems

Cell talks to the view

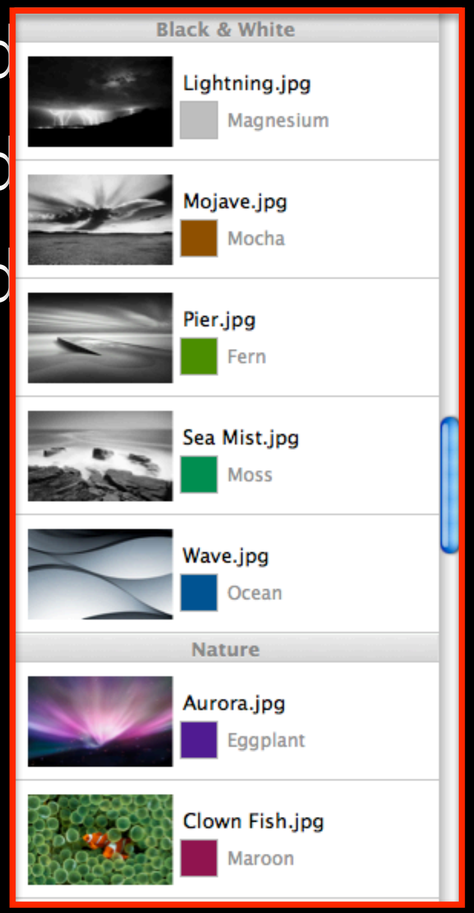



```
[tableView willStartEditingProperty:@"fillColor"
                            forCell:self];
```

How to Solve the Problems

Table view stores information

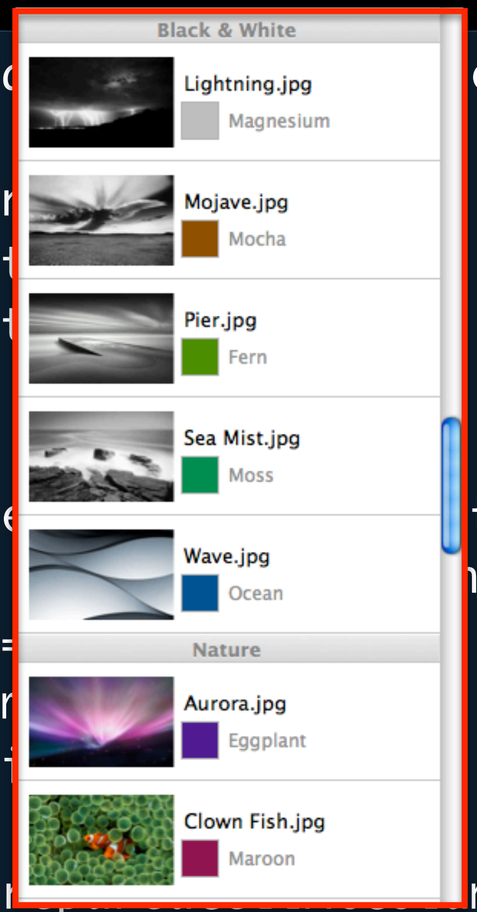
- Ed
- Ed
- Ed



How to Solve the Problems

Table view stores information

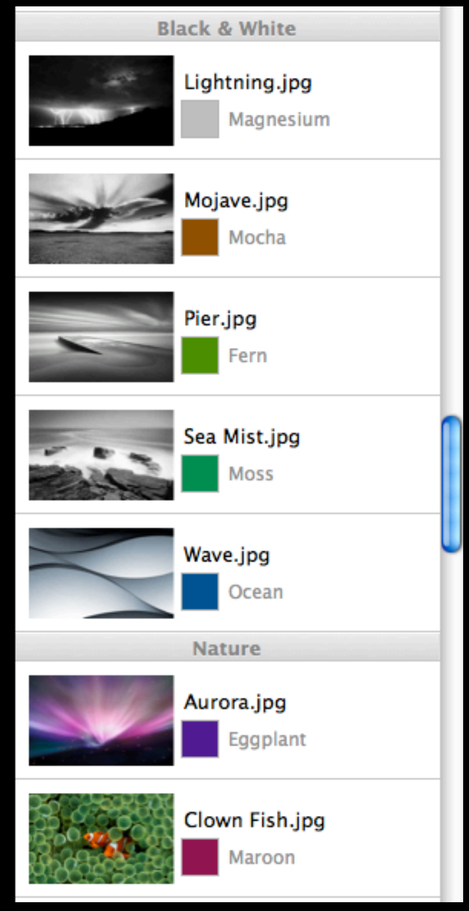
```
@interface ATDynamicTe                                    eView {
@private
    NSCell *_dynEditin
    NSInteger _dynEdit
    NSInteger _dynEdit
}
...
- (NSCell *)preparedCe                                   teger)column
                                                         nteger)row {
  if (_dynEditingRow =
      _dynEditingColu
      return [[_dynEdi                                    autorelease];
  } else {
      return [super pr                                    mn:column
                                                          row:row];
    }
}
```



How to Solve the Problems

Table view stores information

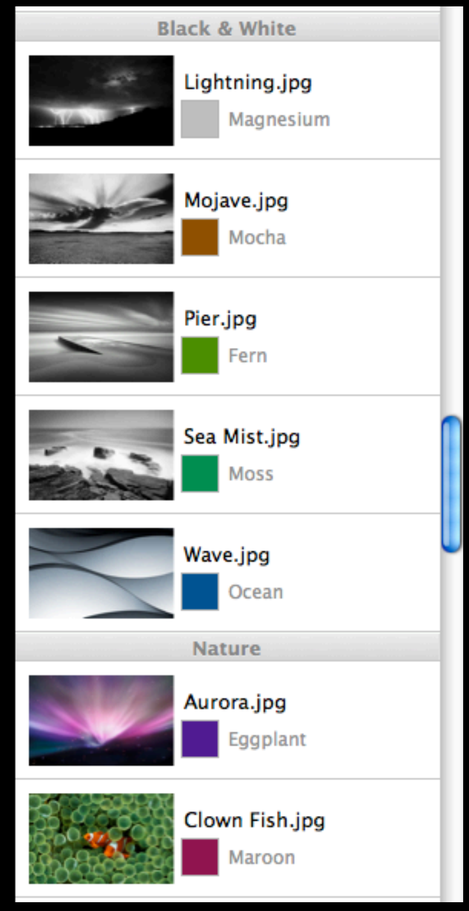
```
@interface ATDynamicTableView : NSTableView {
@private
    NSCell *_dynEditingCell;
    NSInteger _dynEditingRow
    NSInteger _dynEditingColumn;
}
...
- (NSCell *)preparedCellAtColumn:(NSInteger)column
                             row:(NSInteger)row {
  if (_dynEditingRow == row &&
      _dynEditingColumn == column) {
      return [[_dynEditingCell retain] autorelease];
  } else {
      return [super preparedCellAtColumn:column
                                     row:row];
    }
}
```



How to Solve the Problems

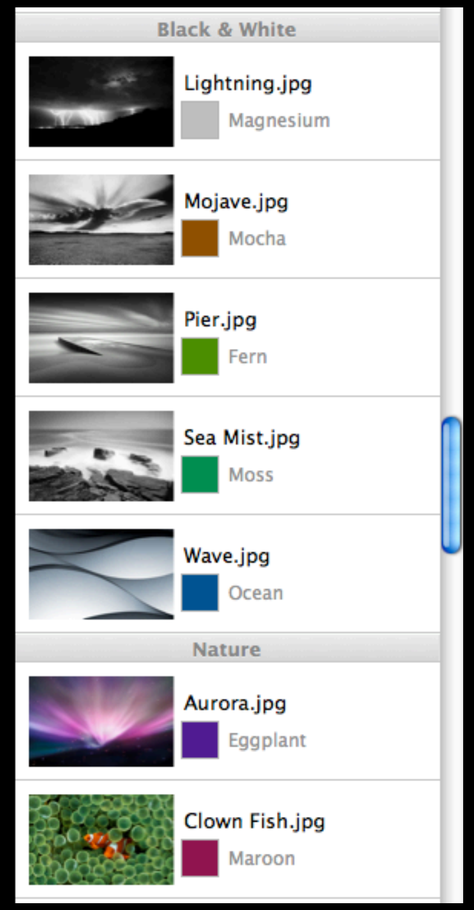
Table view stores information

```
@interface ATDynamicTableView : NSTableView {
@private
    NSCell *_dynEditingCell;
    NSInteger _dynEditingRow
    NSInteger _dynEditingColumn;
}
...
- (NSCell *)preparedCellAtColumn:(NSInteger)column
                             row:(NSInteger)row {
  if (_dynEditingRow == row &&
      _dynEditingColumn == column) {
      return [[_dynEditingCell retain] autorelease];
  } else {
      return [super preparedCellAtColumn:column
                                     row:row];
    }
}
```



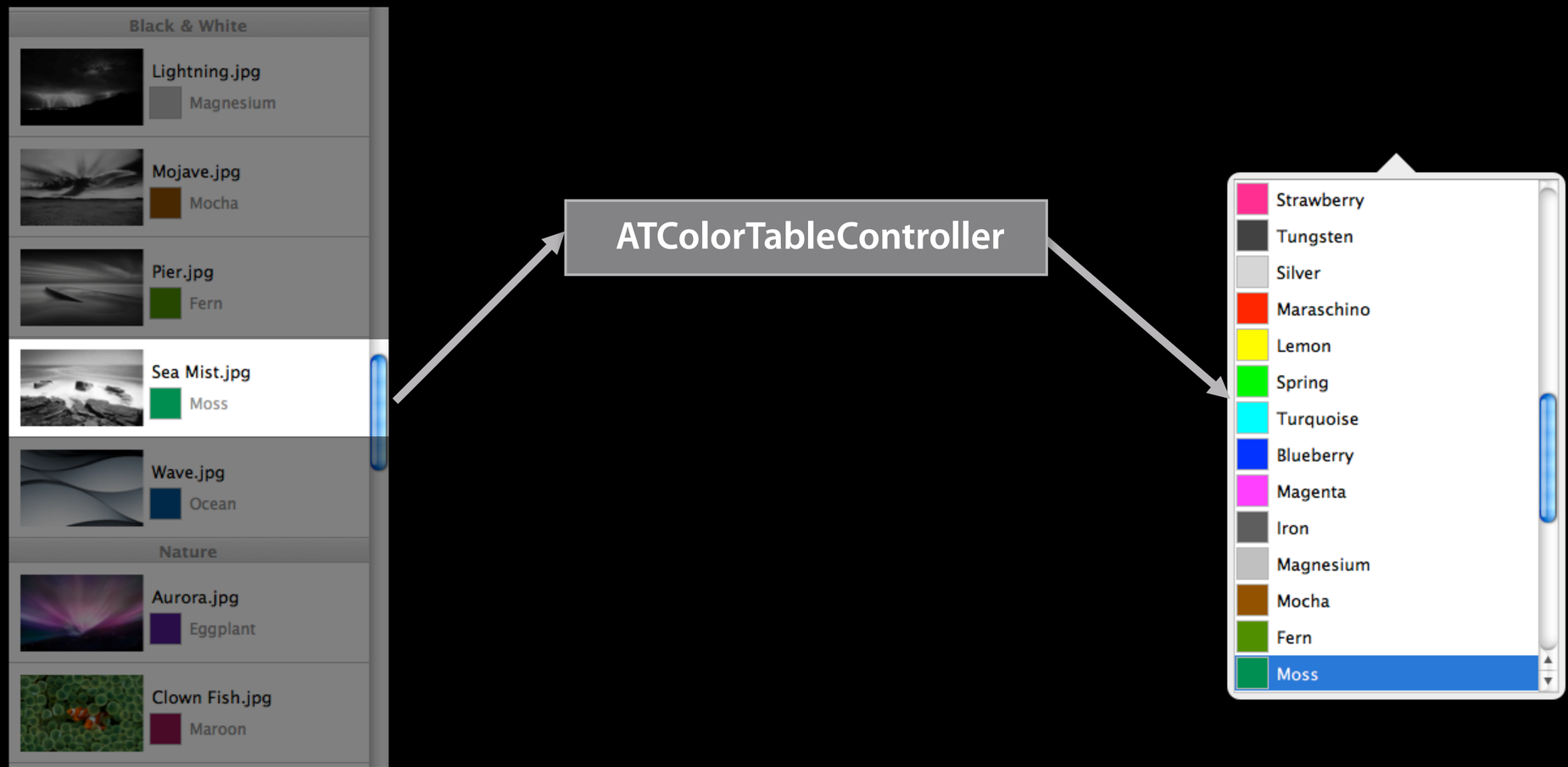
How to Solve the Problems

Cell uses its own controller



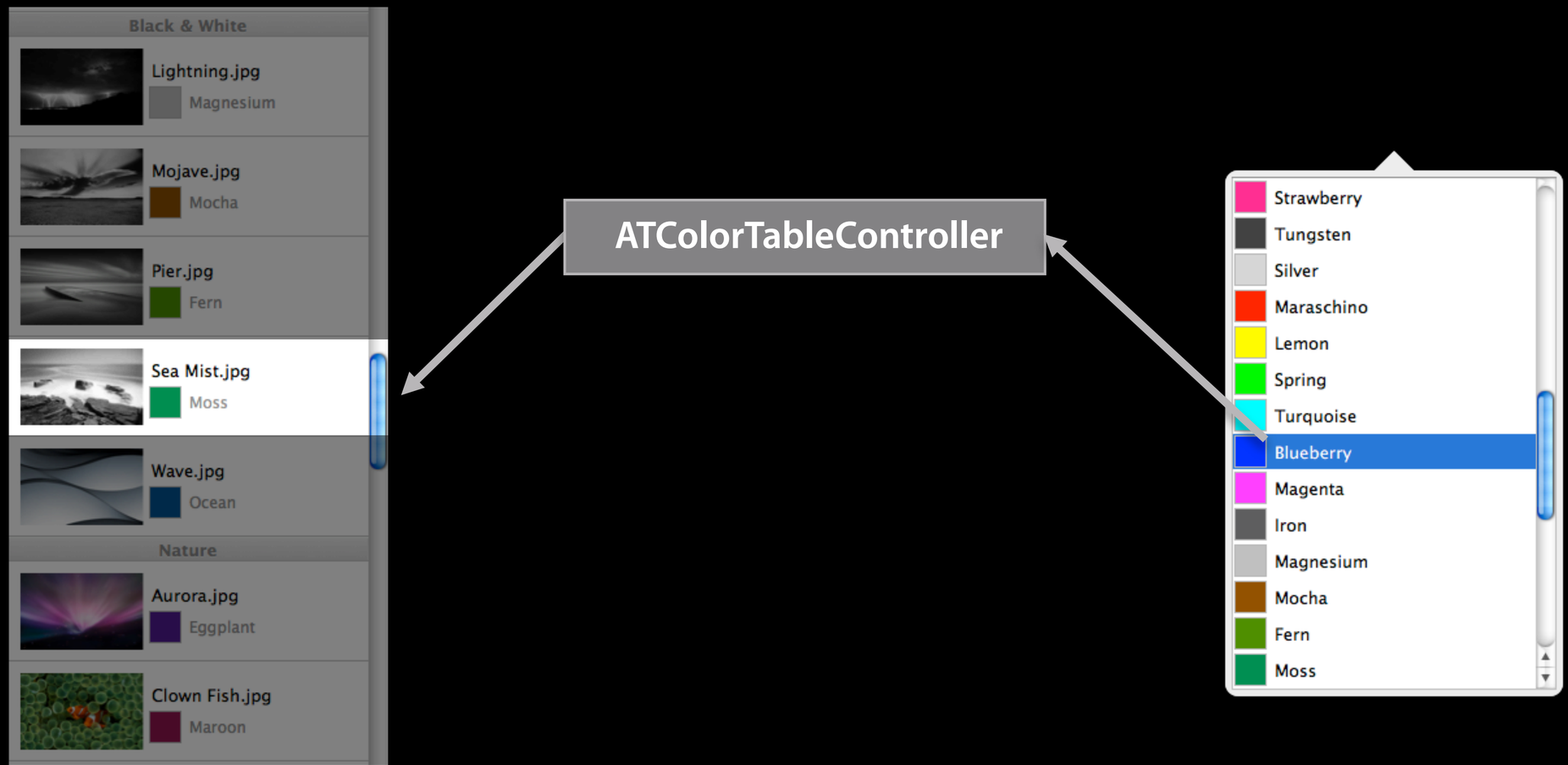
How to Solve the Problems

Cell uses its own controller



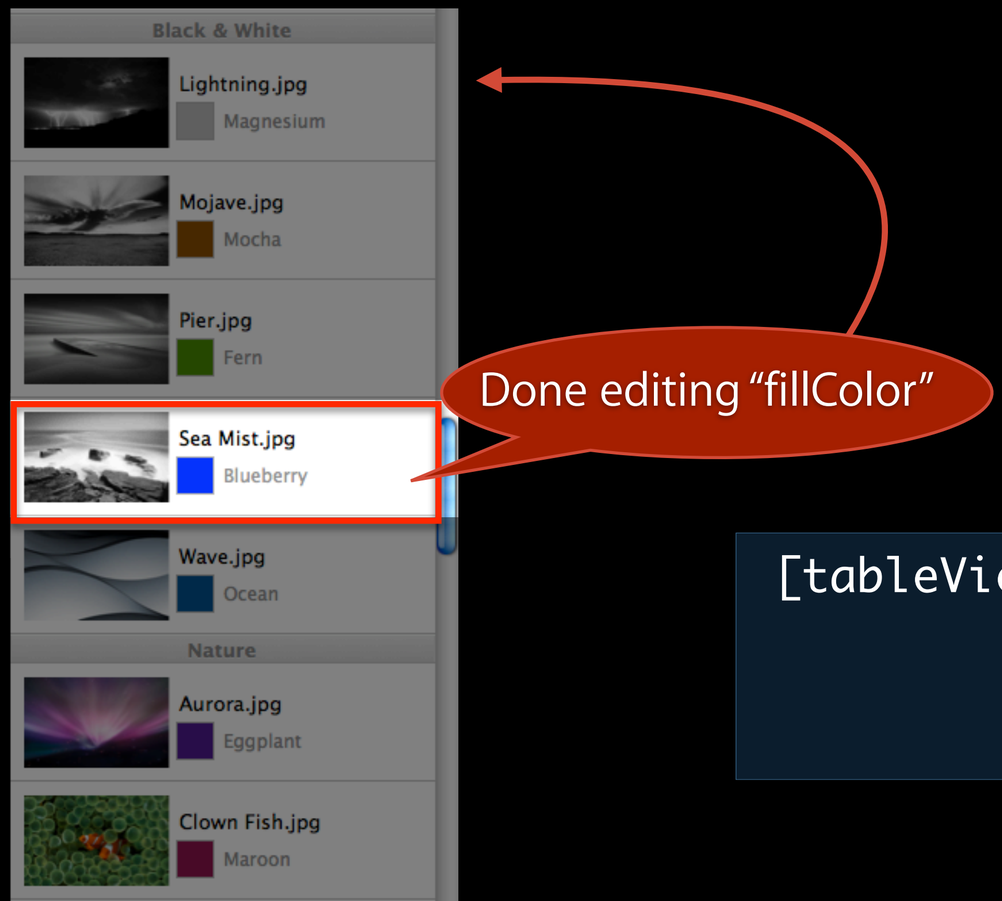
How to Solve the Problems

Cell uses its own controller



How to Solve the Problems

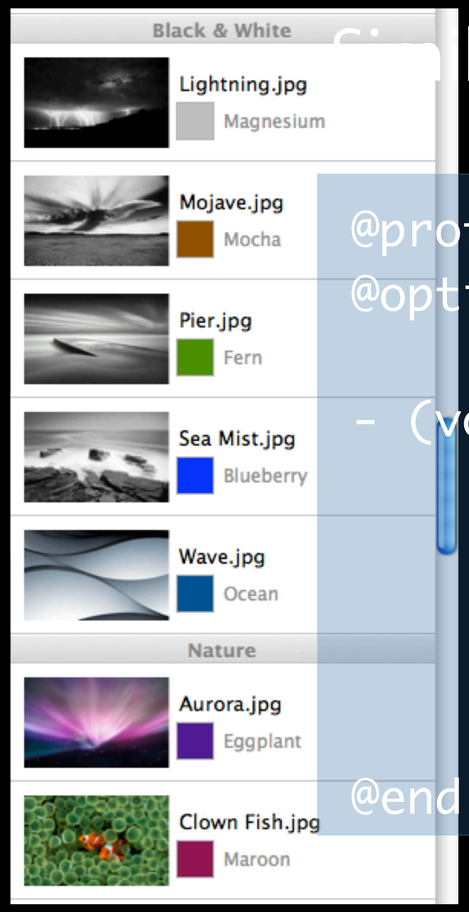
Cell talks to the view




```
[tableView didEndEditingProperty:@"fillColor"
                         forCell:self
                    successfully:YES];
```

How to Solve the Problems

View tells the data source



lar to -tableView:setObjectValue:forTableColumn:row:

```
@protocol ATDynamicTableViewDelegate <NSTableViewDelegate>
@optional

- (void)dynamicTableView:(ATDynamicTableView *)tableView
          setObjectValue:(id)object
          forTableColumn:(NSTableColumn *)tableColumn
                     row:(NSInteger)row
                property:(NSString *)propertyName;

@end
```

How to Solve the Problems

Controller can update the model

```
- (void)dynamicTableView:(ATDynamicTableView *)tableView
          setObjectValue:(id)object
          forTableColumn:(NSTableColumn *)tableColumn
                     row:(NSInteger)row
                property:(NSString *)propertyName {

    ATDesktopImageEntity *entity = [_tableContents objectAtIndex:row];
    if ([propertyName isEqualToString:@"fillColor"]) {
        entity.fillColor = object;
    }
}
```

How to Solve the Problems

Controller can update the model

```
- (void)dynamicTableView:(ATDynamicTableView *)tableView
          setObjectValue:(id)object
          forTableColumn:(NSTableColumn *)tableColumn
                     row:(NSInteger)row
                property:(NSString *)propertyName {

    ATDesktopImageEntity *entity = [_tableContents objectAtIndex:row];
    if ([propertyName isEqualToString:@"fillColor"]) {
        entity.fillColor = object;
    }
}
```

Custom Animations

Animating from NSTableView

What We Are Covering

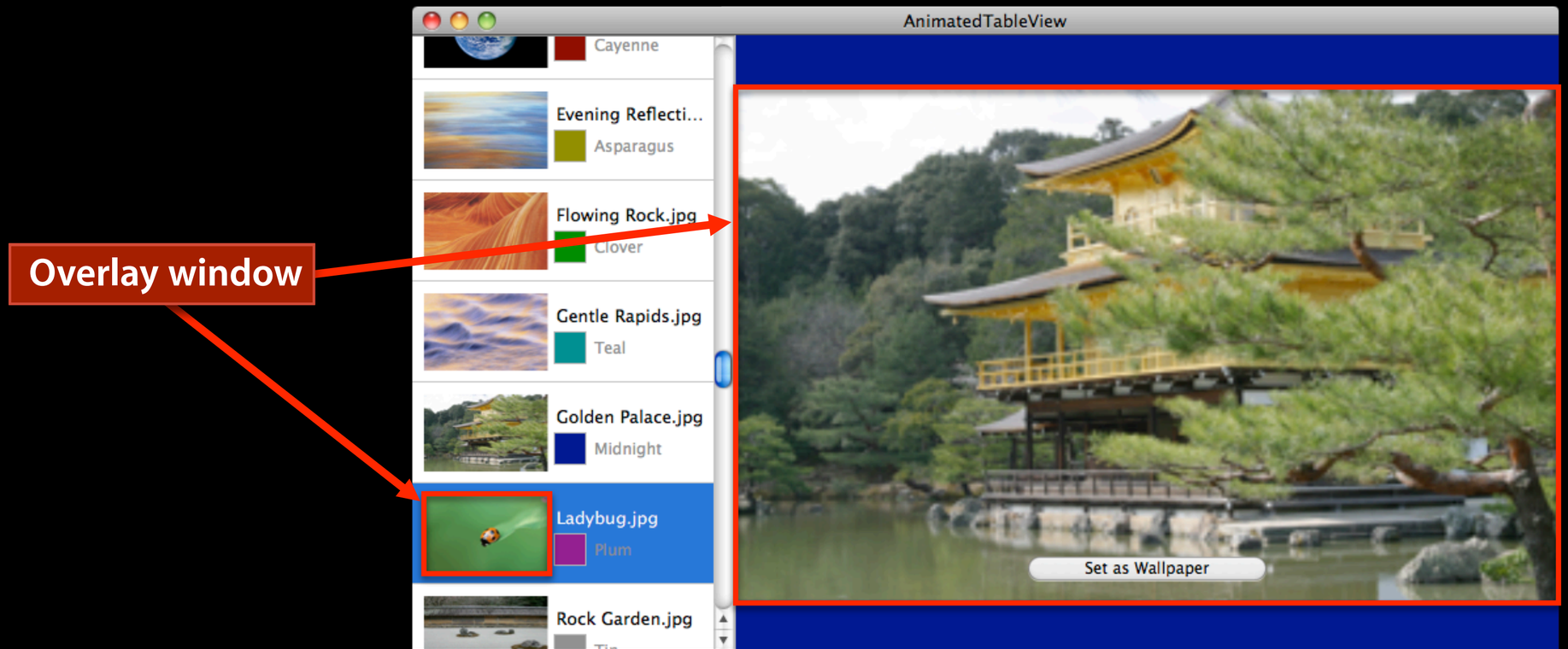
- Subclassing and enhancing NSTableView
  - Batch loading cell contents
  - Adding subviews
  - Custom cell editors
- Custom animations
  - Animating from a table view
  - Animating a pop-up window
- Custom NSBrowsers
  - Utilizing new Snow Leopard APIs

Demo

Animating from a table view

Animating from a Table View

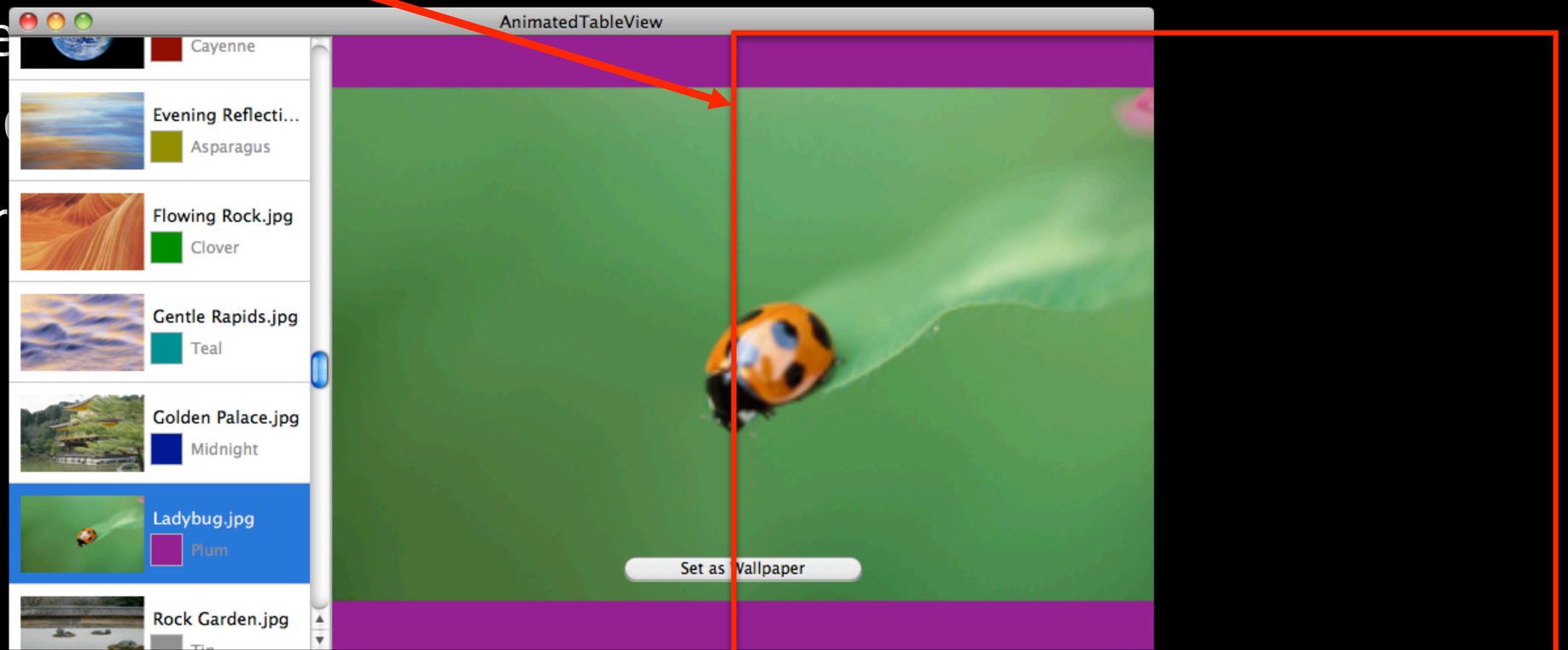
Conceptual outline



Animating from a Table View

Why use an overlay window?

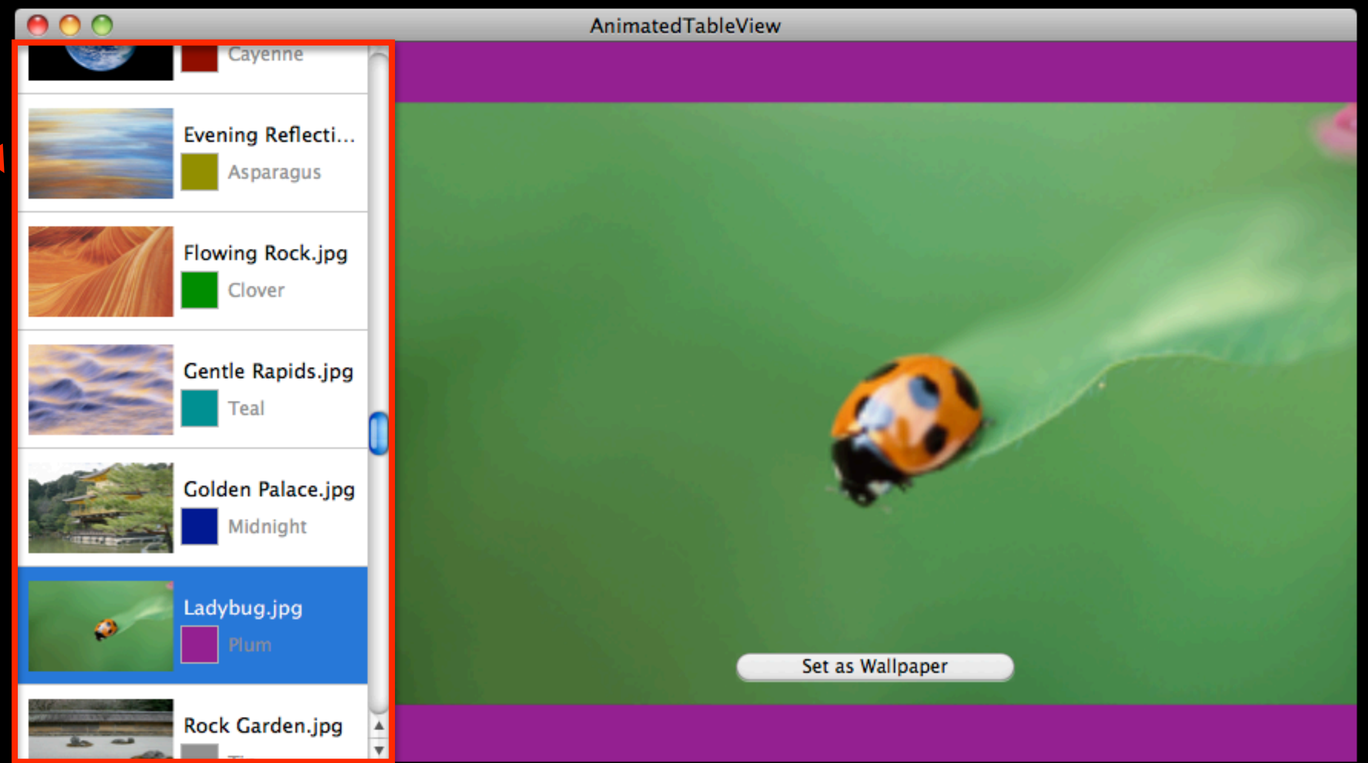
- Layer backed view animations
  - opacity/fade
  - background
- setWantsLayer



Animating from a Table View

Why use an overlay window?

- Nonlayer-backed table view
- Sibling views can't overlap root layer view



Creating a Window for Animating

```
_windowForAnimation = [[NSWindow alloc] initWithContentRect:contentRect
                                                  styleMask:NSBorderlessWindowMask
                                                    backing:NSBackingStoreBuffered
                                                      defer:NO];
[_windowForAnimation setReleasedWhenClosed:NO];
[_windowForAnimation setOpaque:NO];
[_windowForAnimation setBackgroundColor:[NSColor clearColor]];

[[_windowForAnimation contentView] addSubview:_imageViewForTransition];
```

Creating a Window for Animating

```
_windowForAnimation = [[NSWindow alloc] initWithContentRect:contentRect
                                                  styleMask:NSBorderlessWindowMask
                                                    backing:NSBackingStoreBuffered
                                                      defer:NO];
[_windowForAnimation setReleasedWhenClosed:NO];
[_windowForAnimation setOpaque:NO];
[_windowForAnimation setBackgroundColor:[NSColor clearColor]];

[[_windowForAnimation contentView] addSubview:_imageViewForTransition];
```

Initial Window Frame Location

```
- (NSRect)_screenImageRectForRow:(NSInteger)row {
  ATImageTextCell *imageTextCell = [_tableViewMain preparedCellAtColumn:0 row:row];
  NSRect cellFrame = [_tableViewMain frameOfCellAtColumn:0 row:row];
  NSRect cellImageFrame = [imageTextCell imageRectForBounds:cellFrame];

  // Convert to window coordinates
  NSRect result = [_tableViewMain convertRect:cellImageFrame toView:nil];
  result.origin = [_tableViewMain.window convertBaseToScreen:result.origin];
  return result;
}
```

Initial Window Frame Location

```
- (NSRect)_screenImageRectForRow:(NSInteger)row {
  ATImageTextCell *imageTextCell = [_tableViewMain preparedCellAtColumn:0 row:row];
  NSRect cellFrame = [_tableViewMain frameOfCellAtColumn:0 row:row];
  NSRect cellImageFrame = [imageTextCell imageRectForBounds:cellFrame];

  // Convert to window coordinates
  NSRect result = [_tableViewMain convertRect:cellImageFrame toView:nil];
  result.origin = [_tableViewMain.window convertBaseToScreen:result.origin];
  return result;
}
```

Initial Window Frame Location

```
- (NSRect)_screenImageRectForRow:(NSInteger)row {
  ATImageTextCell *imageTextCell = [_tableViewMain preparedCellAtColumn:0 row:row];
  NSRect cellFrame = [_tableViewMain frameOfCellAtColumn:0 row:row];
  NSRect cellImageFrame = [imageTextCell imageRectForBounds:cellFrame];

  // Convert to window coordinates
  NSRect result = [_tableViewMain convertRect:cellImageFrame toView:nil];
  result.origin = [_tableViewMain.window convertBaseToScreen:result.origin];
  return result;
}
```

Initial Window Frame Location

```
- (NSRect)_screenImageRectForRow:(NSInteger)row {
  ATImageTextCell *imageTextCell = [_tableViewMain preparedCellAtColumn:0 row:row];
  NSRect cellFrame = [_tableViewMain frameOfCellAtColumn:0 row:row];
  NSRect cellImageFrame = [imageTextCell imageRectForBounds:cellFrame];

  // Convert to window coordinates
  NSRect result = [_tableViewMain convertRect:cellImageFrame toView:nil];
  result.origin = [_tableViewMain.window convertBaseToScreen:result.origin];
  return result;
}
```

Initial Window Frame Location

```
- (NSRect)_screenImageRectForRow:(NSInteger)row {
  ATImageTextCell *imageTextCell = [_tableViewMain preparedCellAtColumn:0 row:row];
  NSRect cellFrame = [_tableViewMain frameOfCellAtColumn:0 row:row];
  NSRect cellImageFrame = [imageTextCell imageRectForBounds:cellFrame];

  // Convert to window coordinates
  NSRect result = [_tableViewMain convertRect:cellImageFrame toView:nil];
  result.origin = [_tableViewMain.window convertBaseToScreen:result.origin];
  return result;
}
```

Setting Up the Animation

```
[_windowForAnimation orderFront:nil];

[NSAnimationContext beginGrouping];
{
   [[NSAnimationContext currentContext] setDuration:0.4];

   NSRect finalImageFrame = [self _finalScreenImageRect];
   [[_windowForAnimation animator] setFrame:finalImageFrame display:YES];
   [[_imageViewMain animator] setAlphaValue:0.25];
   [[_colorViewMain animator] setBackgroundColor:entity.fillColor];

   _animationDoneTimer = [[NSTimer scheduledTimerWithTimeInterval:0.4
                              target:self
                            selector:@selector(_animationDoneTimerFired:)
                            userInfo:nil repeats:NO] retain];
}
[NSAnimationContext endGrouping];
```

Setting Up the Animation

```
[_windowForAnimation orderFront:nil];

[NSAnimationContext beginGrouping];
{
   [[NSAnimationContext currentContext] setDuration:0.4];

   NSRect finalImageFrame = [self _finalScreenImageRect];
   [[_windowForAnimation animator] setFrame:finalImageFrame display:YES];
   [[_imageViewMain animator] setAlphaValue:0.25];
   [[_colorViewMain animator] setBackgroundColor:entity.fillColor];

   _animationDoneTimer = [[NSTimer scheduledTimerWithTimeInterval:0.4
                              target:self
                            selector:@selector(_animationDoneTimerFired:)
                            userInfo:nil repeats:NO] retain];
}
[NSAnimationContext endGrouping];
```

Setting Up the Animation

```
[_windowForAnimation orderFront:nil];

[NSAnimationContext beginGrouping];
{
   [[NSAnimationContext currentContext] setDuration:0.4];

   NSRect finalImageFrame = [self _finalScreenImageRect];
   [[_windowForAnimation animator] setFrame:finalImageFrame display:YES];
   [[_imageViewMain animator] setAlphaValue:0.25];
   [[_colorViewMain animator] setBackgroundColor:entity.fillColor];

   _animationDoneTimer = [[NSTimer scheduledTimerWithTimeInterval:0.4
                              target:self
                            selector:@selector(_animationDoneTimerFired:)
                            userInfo:nil repeats:NO] retain];
}
[NSAnimationContext endGrouping];
```

Setting Up the Animation

```
[_windowForAnimation orderFront:nil];

[NSAnimationContext beginGrouping];
{
    [[NSAnimationContext currentContext] setDuration:0.4];

    NSRect finalImageFrame = [self _finalScreenImageRect];
    [[_windowForAnimation animator] setFrame:finalImageFrame display:YES];
    [[_imageViewMain animator] setAlphaValue:0.25];
    [[_colorViewMain animator] setBackgroundColor:entity.fillColor];

    _animationDoneTimer = [[NSTimer scheduledTimerWithTimeInterval:0.4
                                target:self
                              selector:@selector(_animationDoneTimerFired:)
                              userInfo:nil repeats:NO] retain];
}
[NSAnimationContext endGrouping];
```

Problem: How can you know when all animations are done?

Setting Up the Animation

```
[_windowForAnimation orderFront:nil];

[NSAnimationContext beginGrouping];
{
   [[NSAnimationContext currentContext] setDuration:0.4];

   NSRect finalImageFrame = [self _finalScreenImageRect];
   [[_windowForAnimation animator] setFrame:finalImageFrame display:YES];
   [[_imageViewMa                                    haValue:0.25];
   [[_colorViewMain animator] setBackgroundColor:entity.fillColor];

   _animationDoneTimer = [[NSTimer scheduledTimerWithTimeInterval:0.4
                                 target:self
                               selector:@selector(_animationDoneTimerFired:)
                               userInfo:nil repeats:NO] retain];
}
[NSAnimationContext endGrouping];
```

Solution: Use a timer.

Clean Up in a Timer

```
- (void)_animationDoneTimerFired:(NSTimer *)timer {
    [_animationDoneTimer release];
    _animationDoneTimer = nil;

    // Set the final image before ordering the window out
    [_imageViewMain setImage:[_imageViewForTransition image]];
    [_imageViewMain setAlphaValue:1.0];
    [_imageViewMain.window displayIfNeeded];

    // Hide the animation window now that the window has drawn
    [_windowForAnimation orderOut:nil];
}
```

Clean Up in a Timer

```
- (void)_animationDoneTimerFired:(NSTimer *)timer {
    [_animationDoneTimer release];
    _animationDoneTimer = nil;

    // Set the final image before ordering the window out
    [_imageViewMain setImage:[_imageViewForTransition image]];
    [_imageViewMain setAlphaValue:1.0];
    [_imageViewMain.window displayIfNeeded];

    // Hide the animation window now that the window has drawn
    [_windowForAnimation orderOut:nil];
}
```

Clean Up in a Timer

```
- (void)_animationDoneTimerFired:(NSTimer *)timer {
    [_animationDoneTimer release];
    _animationDoneTimer = nil;

    // Set the final image before ordering the window out
    [_imageViewMain setImage:[_imageViewForTransition image]];
    [_imageViewMain setAlphaValue:1.0];
    [_imageViewMain.window displayIfNeeded];

    // Hide the animation window now that the window has drawn
    [_windowForAnimation orderOut:nil];
}
```

Clean Up in a Timer

```
- (void)_animationDoneTimerFired:(NSTimer *)timer {
    [_animationDoneTimer release];
    _animationDoneTimer = nil;

    // Set the final image before ordering the window out
    [_imageViewMain setImage:[_imageViewForTransition image]];
    [_imageViewMain setAlphaValue:1.0];
    [_imageViewMain.window displayIfNeeded];

    // Hide the animation window now that the window has drawn
    [_windowForAnimation orderOut:nil];
}
```

Custom Animations

Animating a pop-up window

What We Are Covering

- Subclassing and enhancing NSTableView
  - Batch loading cell contents
  - Adding subviews
  - Custom cell editors
- Custom animations
  - Animating from a table view
  - Animating a pop-up window
- Custom NSBrowsers
  - Utilizing new Snow Leopard APIs

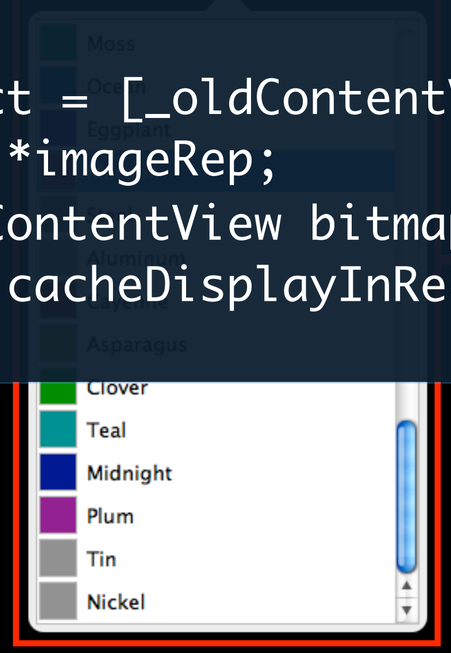
Demo

Animating a pop-up window

Animating a Pop-Up Window

1. Cache the -contentView as an NSBitmapImageRep

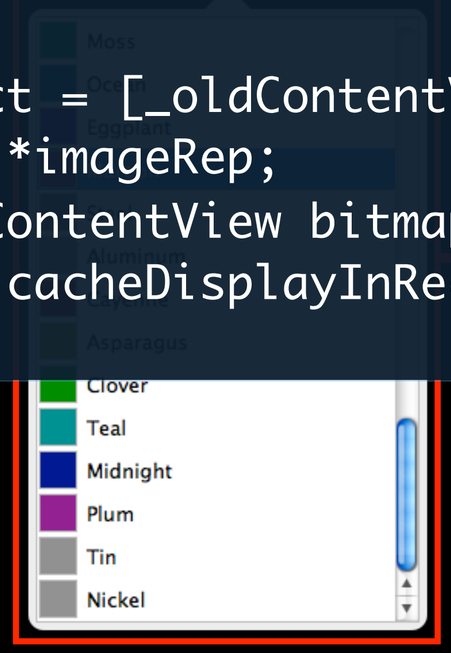
```
// Save the old content view so we can restore it later
_oldContentView = [[self contentView] retain];

NSRect visibleRect = [_oldContentView visibleRect];
NSBitmapImageRep *imageRep;
imageRep = [_oldContentView bitmapImageRepForCachingDisplayInRect:visibleRect];
[_oldContentView cacheDisplayInRect:visibleRect toBitmapImageRep:imageRep];
```



Animating a Pop-Up Window

1. Cache the -contentView as an NSBitmapImageRep

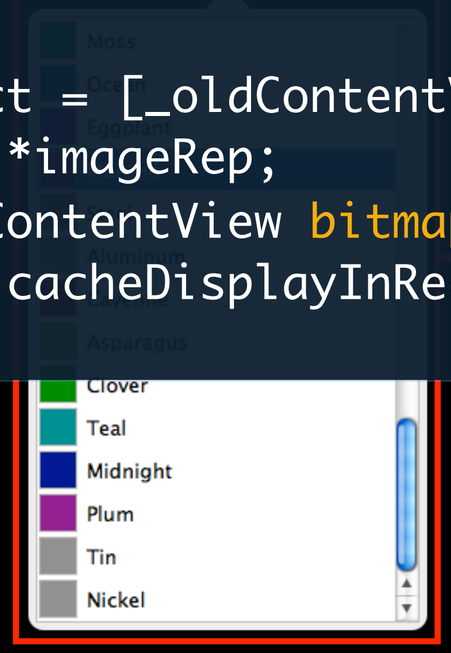
```
// Save the old content view so we can restore it later
_oldContentView = [[self contentView] retain];

NSRect visibleRect = [_oldContentView visibleRect];
NSBitmapImageRep *imageRep;
imageRep = [_oldContentView bitmapImageRepForCachingDisplayInRect:visibleRect];
[_oldContentView cacheDisplayInRect:visibleRect toBitmapImageRep:imageRep];
```



Animating a Pop-Up Window

1. Cache the -contentView as an NSBitmapImageRep

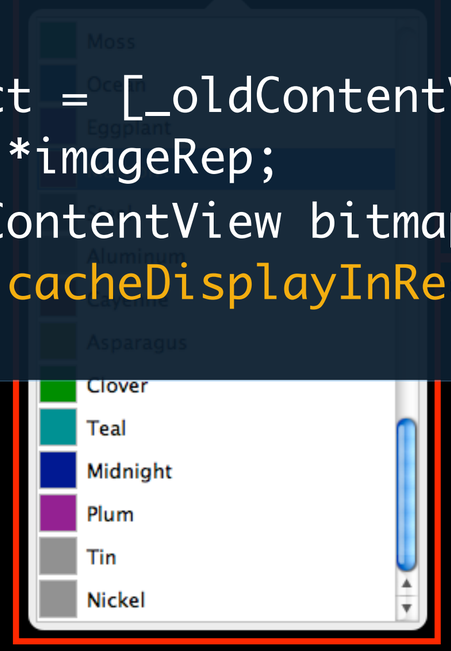
```
// Save the old content view so we can restore it later
_oldContentView = [[self contentView] retain];

NSRect visibleRect = [_oldContentView visibleRect];
NSBitmapImageRep *imageRep;
imageRep = [_oldContentView bitmapImageRepForCachingDisplayInRect:visibleRect];
[_oldContentView cacheDisplayInRect:visibleRect toBitmapImageRep:imageRep];
```



Animating a Pop-Up Window

1. Cache the -contentView as an NSBitmapImageRep

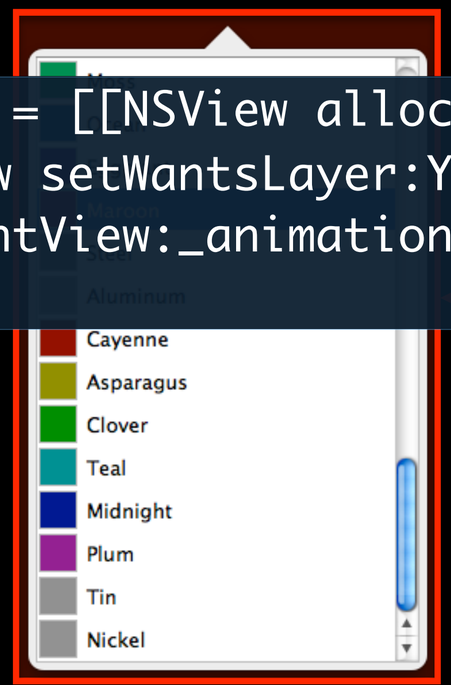
```
// Save the old content view so we can restore it later
_oldContentView = [[self contentView] retain];

NSRect visibleRect = [_oldContentView visibleRect];
NSBitmapImageRep *imageRep;
imageRep = [_oldContentView bitmapImageRepForCachingDisplayInRect:visibleRect];
[_oldContentView cacheDisplayInRect:visibleRect toBitmapImageRep:imageRep];
```



Animating a Pop-Up Window

2. Replace the -contentView with a layer-backed view

```
_animationView = [[NSView alloc] initWithFrame:visibleRect];
[_animationView setWantsLayer:YES];
[self setContentView:_animationView];
```



Layer backed NSView

Animating a Pop-Up Window

2. Replace the -contentView with a layer-backed view
    - Add a child layer for the animation

```
_animationLayer = [CALayer layer];
_animationLayer.frame = NSRectToCGRect(_originalLayerFrame);
_animationLayer.contents = (id)[imageRep CGImage];
_animationLayer.shadowOpacity = 0.50;
_animationLayer.shadowRadius = 4;
// Start at 1% scale
_animationLayer.transform = [self _transformForScale:0.01];

// Get the layer into the rendering tree
[[_animationView layer] addSublayer:_animationLayer];
```

Animating a Pop-Up Window

2. Replace the -contentView with a layer-backed view
    - Add a child layer for the animation

```
_animationLayer = [CALayer layer];
_animationLayer.frame = NSRectToCGRect(_originalLayerFrame);
_animationLayer.contents = (id)[imageRep CGImage];
_animationLayer.shadowOpacity = 0.50;
_animationLayer.shadowRadius = 4;
// Start at 1% scale
_animationLayer.transform = [self _transformForScale:0.01];

// Get the layer into the rendering tree
[[_animationView layer] addSublayer:_animationLayer];
```

Animating a Pop-Up Window

2. Replace the -contentView with a layer-backed view
    - Add a child layer for the animation

```
_animationLayer = [CALayer layer];
_animationLayer.frame = NSRectToCGRect(_originalLayerFrame);
_animationLayer.contents = (id)[imageRep CGImage];
_animationLayer.shadowOpacity = 0.50;
_animationLayer.shadowRadius = 4;
// Start at 1% scale
_animationLayer.transform = [self _transformForScale:0.01];

// Get the layer into the rendering tree
[[_animationView layer] addSublayer:_animationLayer];
```

Animating a Pop-Up Window

3. Use CoreAnimation to animate the layer's transform

```
- (void)_addAnimationToScale:(CGFloat)scale duration:(NSTimeInterval)duration {
  CABasicAnimation *transformAni = [CABasicAnimation animation];
  // We are animating from the current transform to the new value requested
  transformAni.fromValue =
                  [NSValue valueWithCATransform3D:_animationLayer.transform];
  transformAni.duration = duration;
  transformAni.delegate = self;
  // The final value for the animation will the the current value the layer has
  _animationLayer.transform = [self _transformForScale:scale];
  [_animationLayer addAnimation:transformAni forKey:@"transform"];
}
```

Animating a Pop-Up Window

3. Use CoreAnimation to animate the layer's transform

```
- (void)_addAnimationToScale:(CGFloat)scale duration:(NSTimeInterval)duration {
  CABasicAnimation *transformAni = [CABasicAnimation animation];
  // We are animating from the current transform to the new value requested
  transformAni.fromValue =
                  [NSValue valueWithCATransform3D:_animationLayer.transform];
  transformAni.duration = duration;
  transformAni.delegate = self;
  // The final value for the animation will the the current value the layer has
  _animationLayer.transform = [self _transformForScale:scale];
  [_animationLayer addAnimation:transformAni forKey:@"transform"];
}
```

Animating a Pop-Up Window

3. Use CoreAnimation to animate the layer's transform

```
- (void)_addAnimationToScale:(CGFloat)scale duration:(NSTimeInterval)duration {
  CABasicAnimation *transformAni = [CABasicAnimation animation];
  // We are animating from the current transform to the new value requested
  transformAni.fromValue =
                    [NSValue valueWithCATransform3D:_animationLayer.transform];
  transformAni.duration = duration;
  transformAni.delegate = self;
  // The final value for the animation will the the current value the layer has
  _animationLayer.transform = [self _transformForScale:scale];
  [_animationLayer addAnimation:transformAni forKey:@"transform"];
}
```

Animating a Pop-Up Window

3. Use CoreAnimation to animate the layer's transform

```
- (void)_addAnimationToScale:(CGFloat)scale duration:(NSTimeInterval)duration {
  CABasicAnimation *transformAni = [CABasicAnimation animation];
  // We are animating from the current transform to the new value requested
  transformAni.fromValue =
                  [NSValue valueWithCATransform3D:_animationLayer.transform];
  transformAni.duration = duration;
  transformAni.delegate = self;
  // The final value for the animation will the the current value the layer has
  _animationLayer.transform = [self _transformForScale:scale];
  [_animationLayer addAnimation:transformAni forKey:@"transform"];
}
```

Animating a Pop-Up Window

3. Use CoreAnimation to animate the layer's transform

```
- (void)_addAnimationToScale:(CGFloat)scale duration:(NSTimeInterval)duration {
  CABasicAnimation *transformAni = [CABasicAnimation animation];
  // We are animating from the current transform to the new value requested
  transformAni.fromValue =
                  [NSValue valueWithCATransform3D:_animationLayer.transform];
  transformAni.duration = duration;
  transformAni.delegate = self;
  // The final value for the animation will the the current value the layer has
  _animationLayer.transform = [self _transformForScale:scale];
  [_animationLayer addAnimation:transformAni forKey:@"transform"];
}
```

Animating a Pop-Up Window

3. Use CoreAnimation to animate the layer's transform

```
- (void)_addAnimationToScale:(CGFloat)scale duration:(NSTimeInterval)duration {
  CABasicAnimation *transformAni = [CABasicAnimation animation];
  // We are animating from the current transform to the new value requested
  transformAni.fromValue =
                  [NSValue valueWithCATransform3D:_animationLayer.transform];
  transformAni.duration = duration;
  transformAni.delegate = self;
  // The final value for the animation will the the current value the layer has
  _animationLayer.transform = [self _transformForScale:scale];
  [_animationLayer addAnimation:transformAni forKey:@"transform"];
}
```

Animating a Pop-Up Window

3. Use CoreAnimation to animate the layer's transform
    - Chain multiple animations for a "pop" effect

```
- (void)animationDidStop:(CAAnimation *)anim finished:(BOOL)flag {
  if (_growing) {
    _growing = NO;
    _shrinking = YES;
    [self _addAnimationToScale:SHRINK_SCALE duration:SHRINK_ANIMATION_DURATION];
  } else if (_shrinking) {
    _shrinking = NO;
    [self _addAnimationToScale:1.0 duration:RESTORE_ANIMATION_DURATION];
  } else {
    [self _cleanupAndRestoreViews];
  }
}
```

Animating a Pop-Up Window

3. Use CoreAnimation to animate the layer's transform
    - Chain multiple animations for a "pop" effect

```
- (void)animationDidStop:(CAAnimation *)anim finished:(BOOL)flag {
  if (_growing) {
    _growing = NO;
    _shrinking = YES;
    [self _addAnimationToScale:SHRINK_SCALE duration:SHRINK_ANIMATION_DURATION];
  } else if (_shrinking) {
    _shrinking = NO;
    [self _addAnimationToScale:1.0 duration:RESTORE_ANIMATION_DURATION];
  } else {
    [self _cleanupAndRestoreViews];
  }
}
```

Animating a Pop-Up Window

3. Use CoreAnimation to animate the layer's transform
    - Chain multiple animations for a "pop" effect

```
- (void)animationDidStop:(CAAnimation *)anim finished:(BOOL)flag {
  if (_growing) {
    _growing = NO;
    _shrinking = YES;
    [self _addAnimationToScale:SHRINK_SCALE duration:SHRINK_ANIMATION_DURATION];
  } else if (_shrinking) {
    _shrinking = NO;
    [self _addAnimationToScale:1.0 duration:RESTORE_ANIMATION_DURATION];
  } else {
    [self _cleanupAndRestoreViews];
  }
}
```

Animating a Pop-Up Window

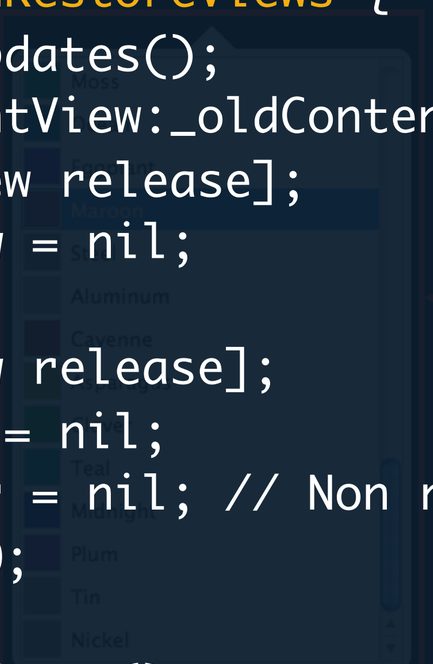
3. Use CoreAnimation to animate the layer's transform
    - Chain multiple animations for a "pop" effect

```
- (void)animationDidStop:(CAAnimation *)anim finished:(BOOL)flag {
  if (_growing) {
    _growing = NO;
    _shrinking = YES;
    [self _addAnimationToScale:SHRINK_SCALE duration:SHRINK_ANIMATION_DURATION];
  } else if (_shrinking) {
    _shrinking = NO;
    [self _addAnimationToScale:1.0 duration:RESTORE_ANIMATION_DURATION];
  } else {
    [self _cleanupAndRestoreViews];
  }
}
```

Animating a Pop-Up Window

4. Restore the original -contentView at the end of the animation

```
- (void)_cleanupAndRestoreViews {
  NSDisableScreenUpdates();
    [self setContentView:_oldContentView];
    [_oldContentView release];
    _oldContentView = nil;

    [_animationView release];
    _animationView = nil;
    _animationLayer = nil; // Non retained
    _shrinking = NO;
    _growing = NO;
  NSEnableScreenUpdates();
}
```



Original -contentView

Custom NSBrowsers

Snow Leopard API, Headers, and Previews

Raleigh Ledet
Cocoa Software Engineer

What We Are Covering

- Subclassing and enhancing NSTableView
  - Batch loading cell contents
  - Adding subviews
  - Custom cell editors
- Custom animations
  - Animating from a table view
  - Animating a pop-up window
- Custom NSBrowsers
  - Utilizing new Snow Leopard APIs

Demo

Snow Leopard header and preview views

Custom NSBrowsers

- Item-based browsers
- Custom header views
- Custom preview views

Item-Based NSBrowser

- API is similar to NSOutlineView
  - But, items do NOT have to be unique
- Implementation no longer uses NSMatrix
  - Matrix-based methods no longer applicable

```
- (void)setMatrixClass:(Class)factoryId;
- (Class)matrixClass
- (NSInteger)columnOfMatrix:(NSMatrix *)matrix;
- (NSMatrix *)matrixInColumn:(NSInteger)column;
```

- Many new Snow Leopard methods only work with the new API

Item-Based NSBrowser

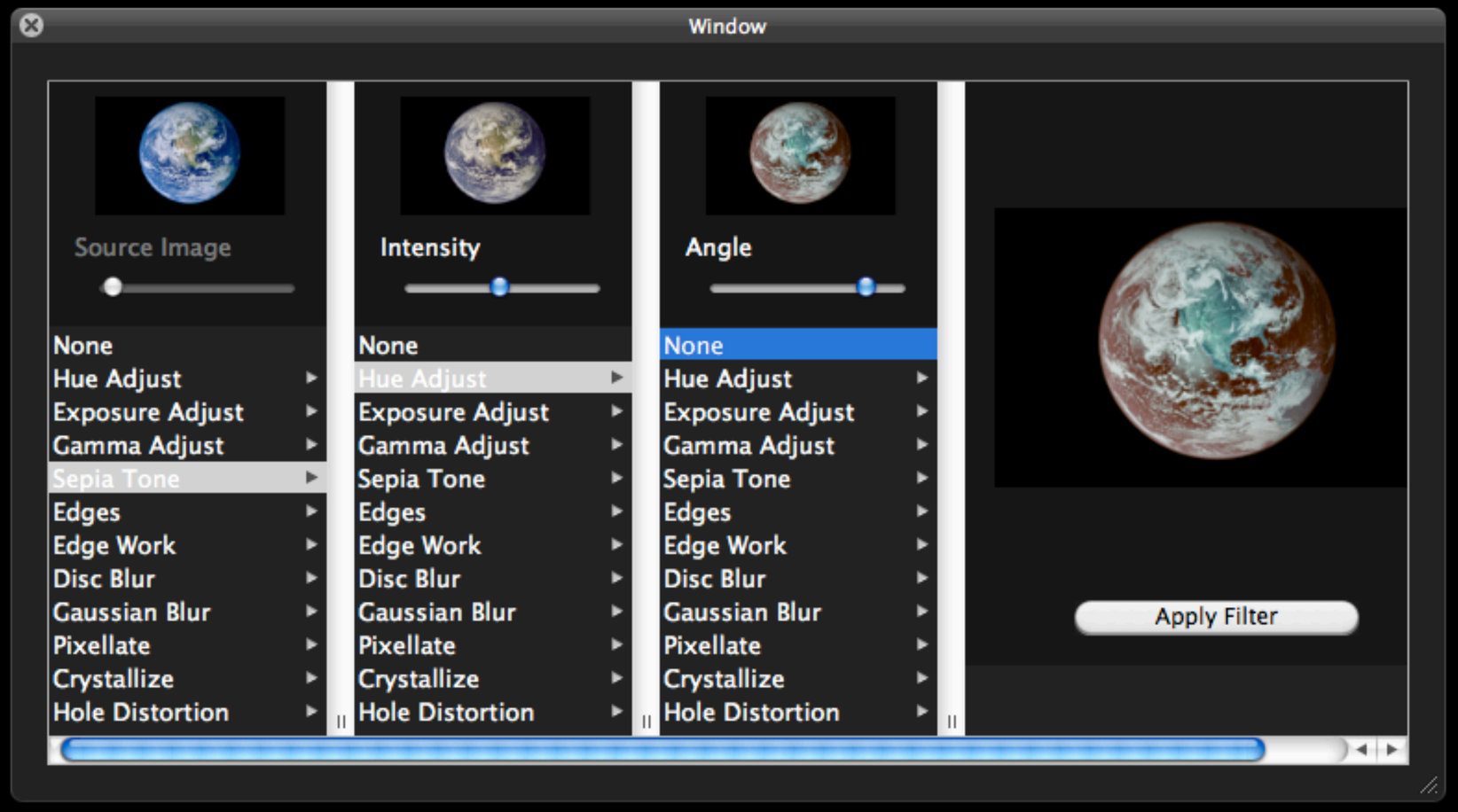
Custom cells

- Existing API still works

```
- (void)setCellClass:(Class)factoryId;
- (void)setCellPrototype:(NSCell *)aCell;
- (id)cellPrototype;


- (void)browser:(NSBrowser *)sender willDisplayCell:(id)cell
                                                 atRow:(NSInteger)row
                                                column:(NSInteger)column;
```

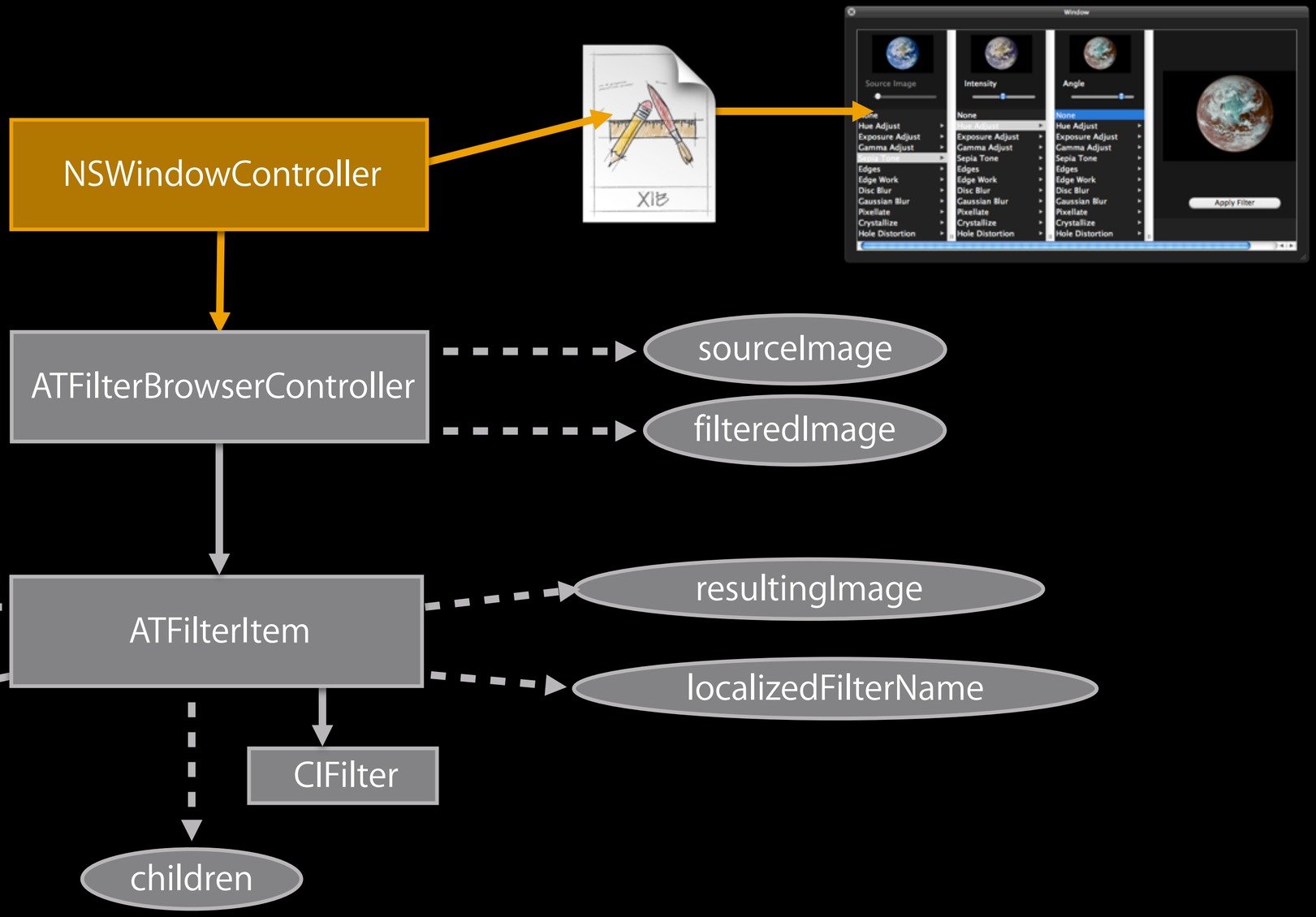
Item-Based NSBrowser

Model—Class hierarchy



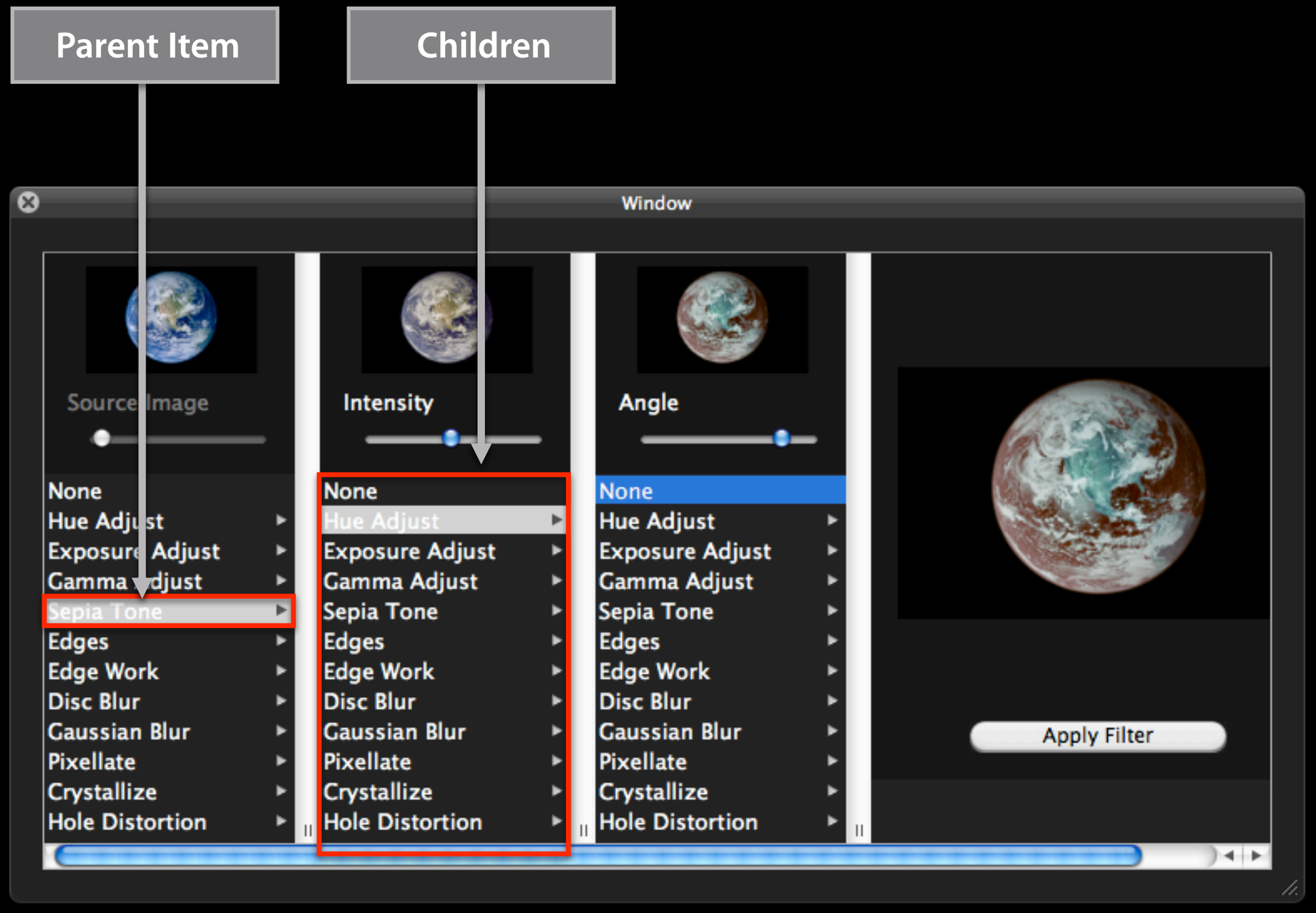
Item-Based NSBrowser

Model—Class hierarchy



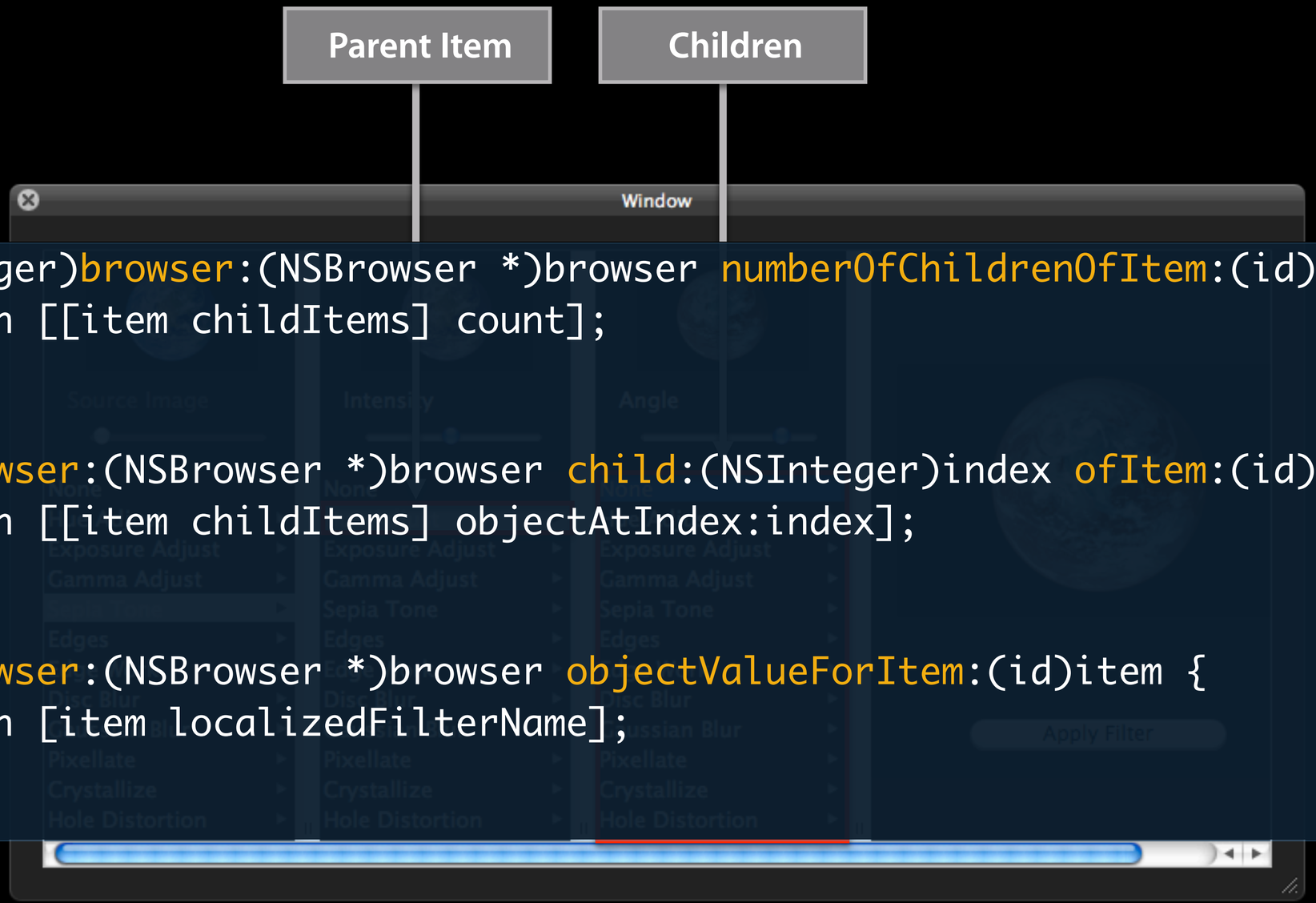
Item-Based NSBrowser

Delegate methods



Item-Based NSBrowser

Delegate methods




```
- (NSInteger)browser:(NSBrowser *)browser numberOfChildrenOfItem:(id)item {
    return [[item childItems] count];
}

- (id)browser:(NSBrowser *)browser child:(NSInteger)index ofItem:(id)item  {
    return [[item childItems] objectAtIndex:index];
}

- (id)browser:(NSBrowser *)browser objectValueForItem:(id)item {
    return [item localizedFilterName];
}
```

Item-Based NSBrowser

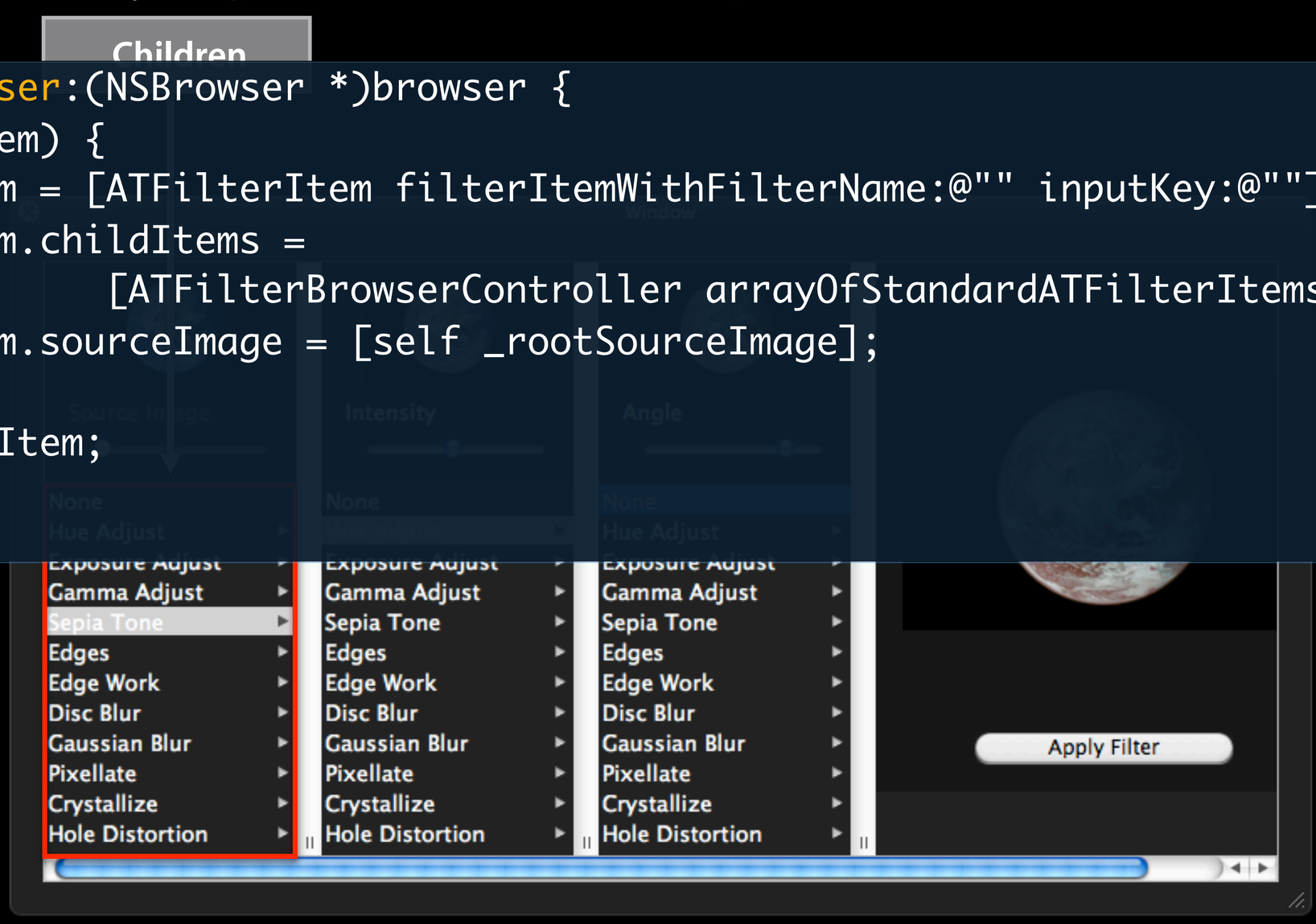
Specifying the root item (optional)

Parent Item

Children

```
- (id)rootItemForBrowser:(NSBrowser *)browser {
  if (!_rootBrowserItem) {
      _rootBrowserItem = [ATFilterItem filterItemWithFilterName:@"" inputKey:@""];
      _rootBrowserItem.childItems =
                          [ATFilterBrowserController arrayOfStandardATFilterItems];
      _rootBrowserItem.sourceImage = [self _rootSourceImage];
  }
  return _rootBrowserItem;
}
```

_rootBrowserItem

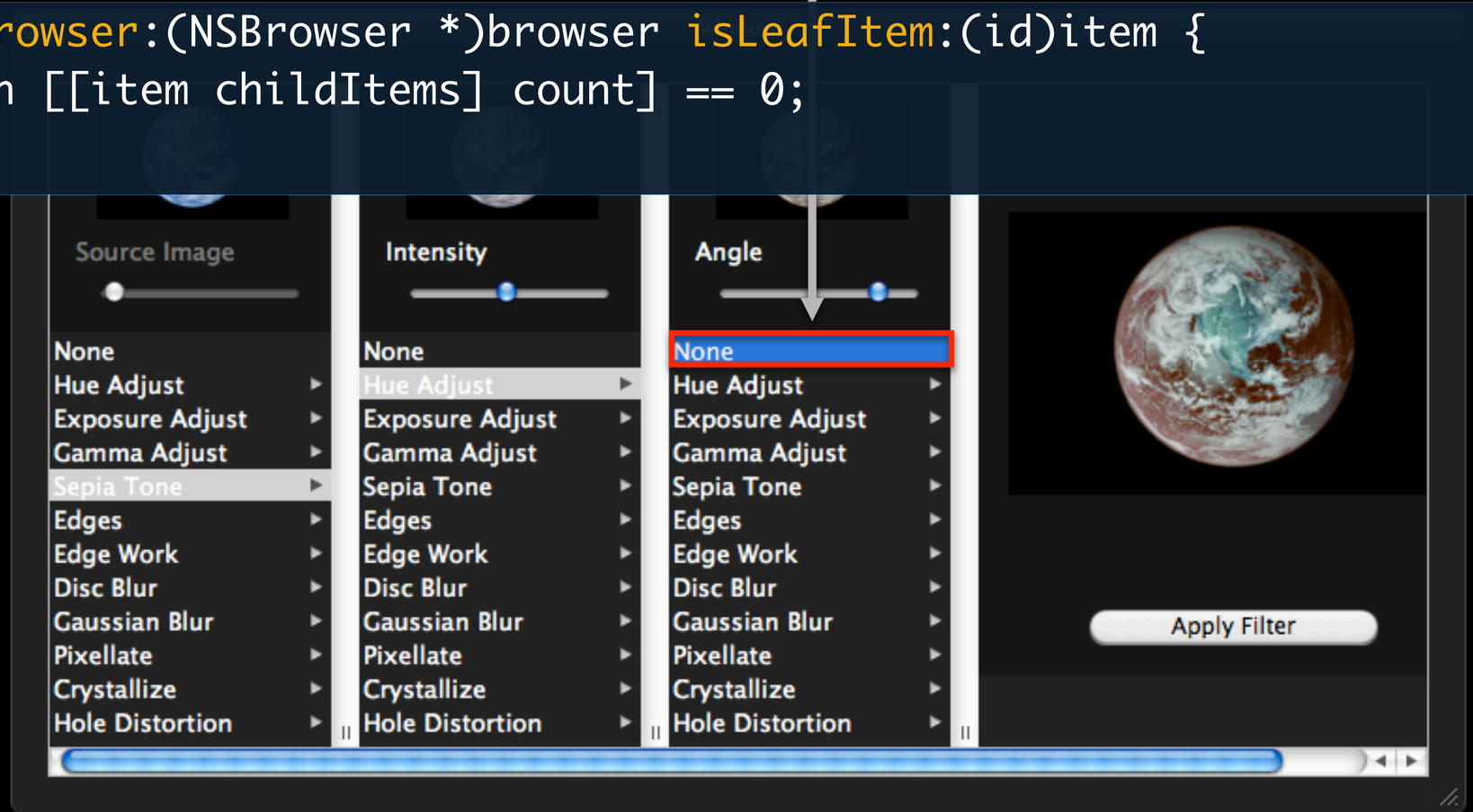


Item-Based NSBrowser

Delegate methods

Leaf Item

```
- (BOOL)browser:(NSBrowser *)browser isLeafItem:(id)item {
    return [[item childItems] count] == 0;
}
```



Item-Based NSBrowser

Useful methods—Column number to parent item

```
- (void)browser:(NSBrowser *)browser
    didChangeLastColumn:(NSInteger)oldLastColumn
    toColumn:(NSInteger)column {

    if (column > 0) {
        ATFilterItem *item = [browser parentForItemsInColumn:column];
        [item bind:@"sourceImage"
            toObject:[browser parentForItemsInColumn:column - 1]
            withKeyPath:@"resultingImage" options:nil];
    }
}
```

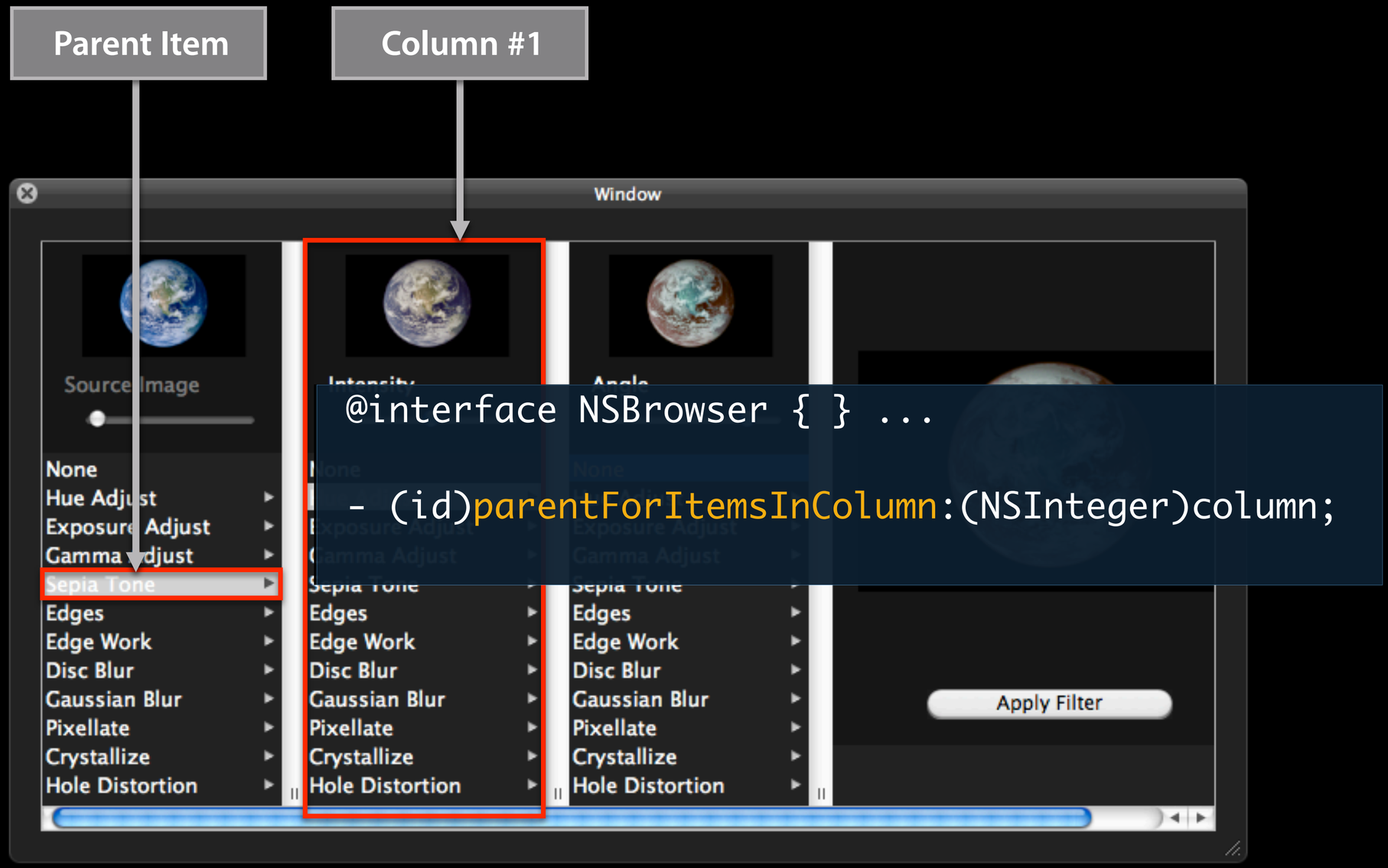
Item-Based NSBrowser

Useful methods—Column number to parent item

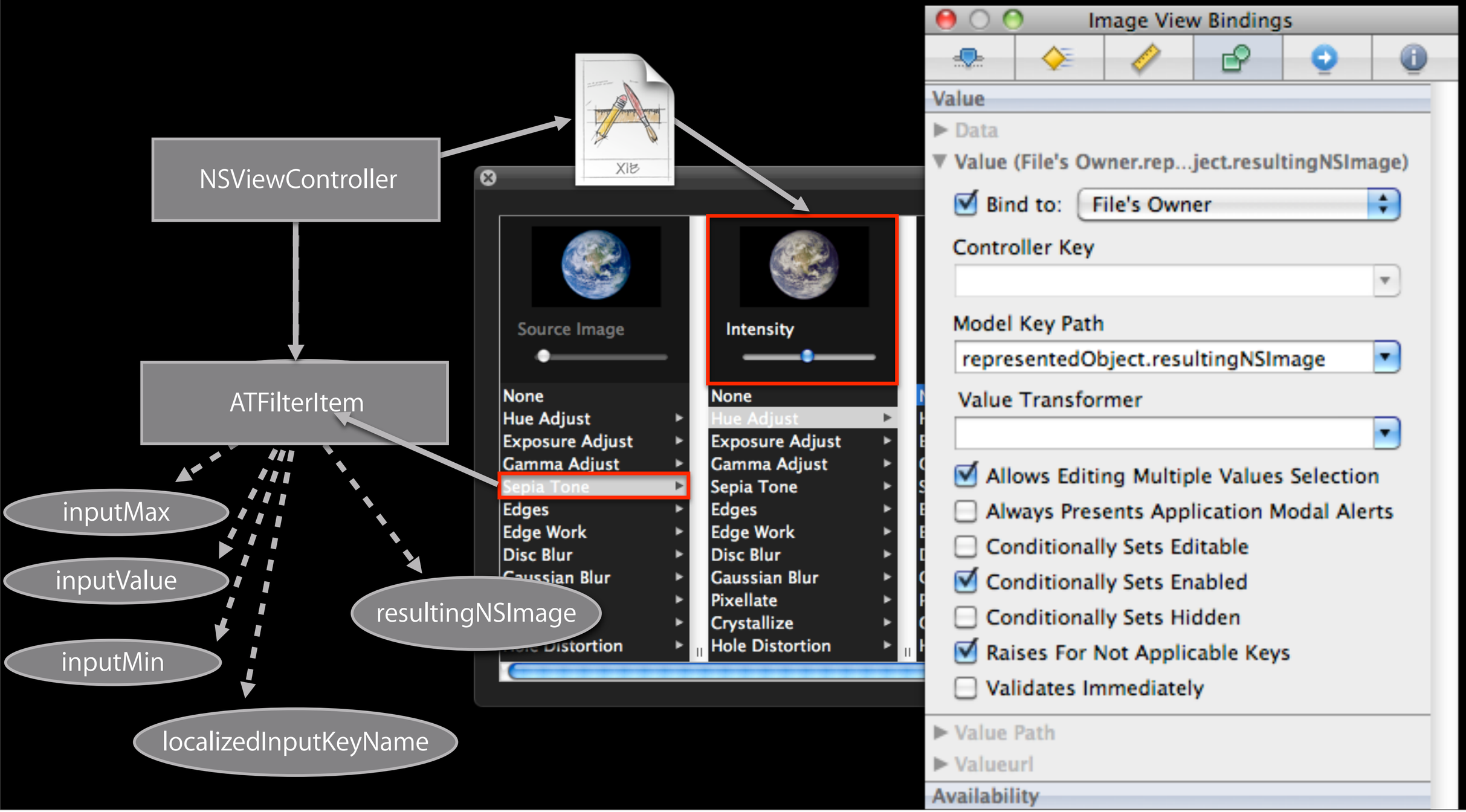
```
- (void)browser:(NSBrowser *)browser
    didChangeLastColumn:(NSInteger)oldLastColumn
    toColumn:(NSInteger)column {

    if (column > 0) {
        ATFilterItem *item = [browser parentForItemsInColumn:column];
        [item bind:@"sourceImage"
            toObject:[browser parentForItemsInColumn:column - 1]
            withKeyPath:@"resultingImage" options:nil];
    }
}
```

Item-Based NSBrowser

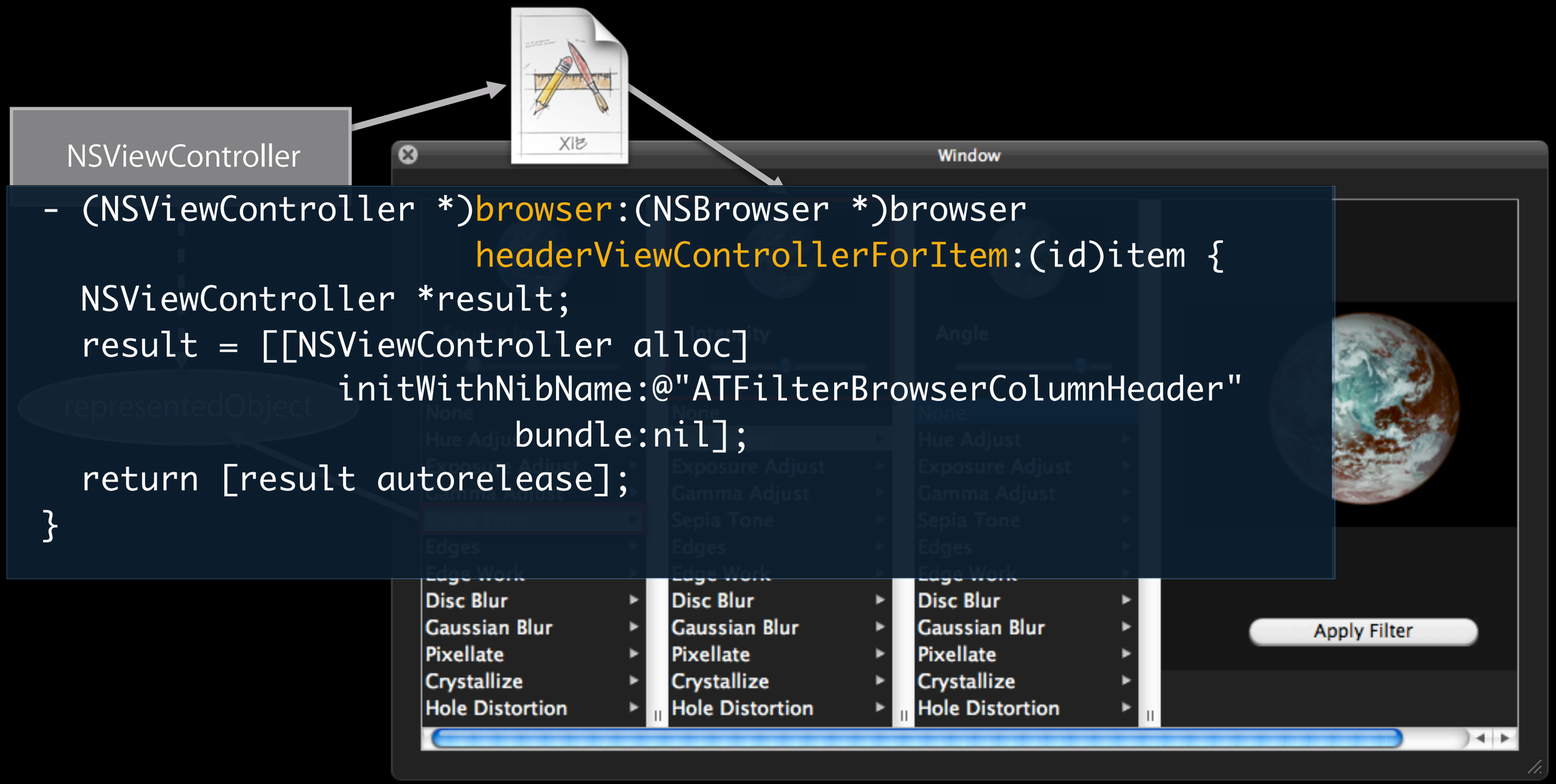
Useful methods—Column number to parent item



Custom Header Views



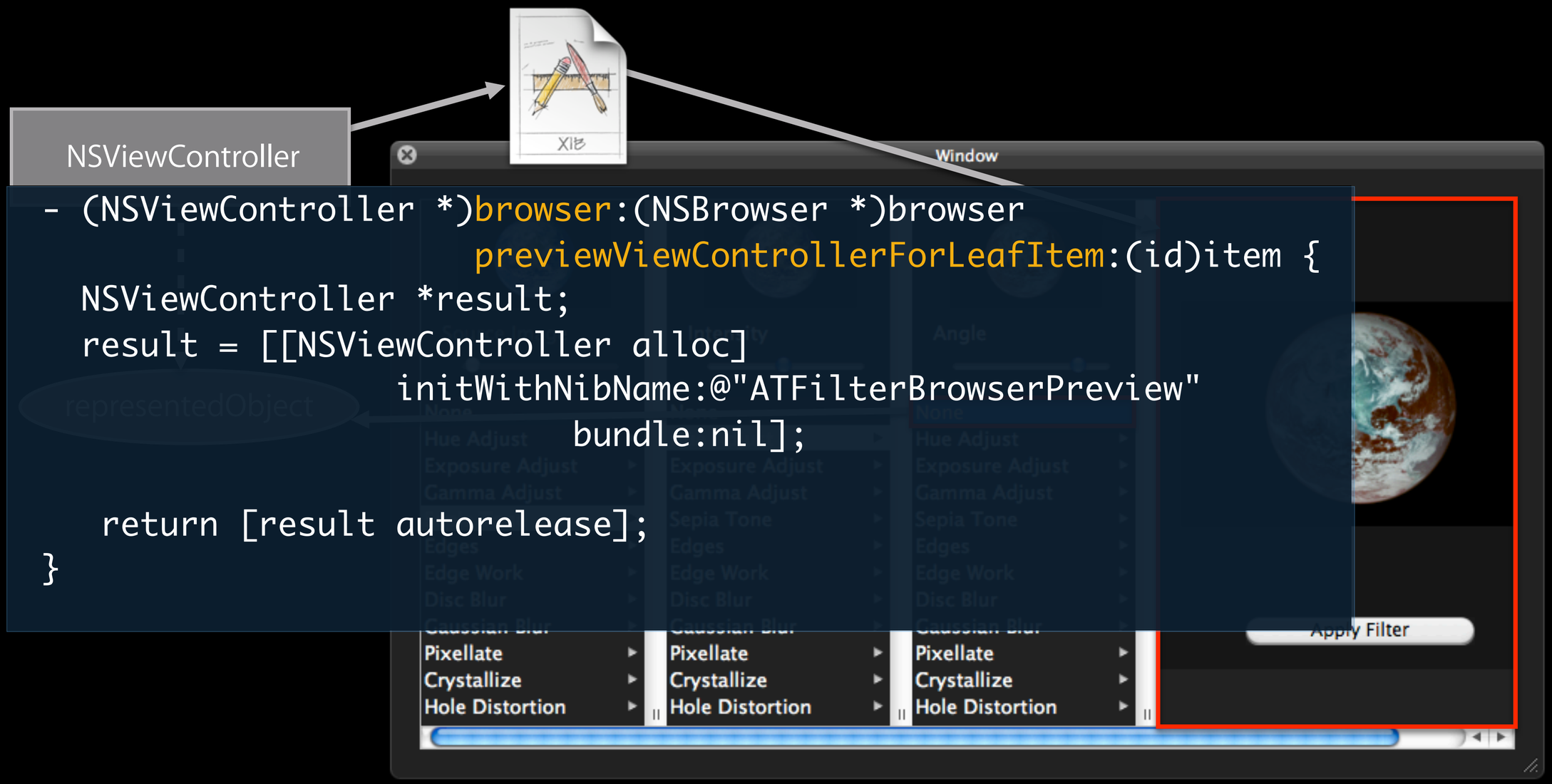
Custom Header Views




```
- (NSViewController *)browser:(NSBrowser *)browser
                 headerViewControllerForItem:(id)item {
  NSViewController *result;
  result = [[NSViewController alloc]
                initWithNibName:@"ATFilterBrowserColumnHeader"
                         bundle:nil];
  return [result autorelease];
}
```

Custom Preview Views

Very similar to header views




```
- (NSViewController *)browser:(NSBrowser *)browser
                 previewViewControllerForLeafItem:(id)item {
  NSViewController *result;
  result = [[NSViewController alloc]
                initWithNibName:@"ATFilterBrowserPreview"
                         bundle:nil];

   return [result autorelease];
}
```

Item-Based NSBrowser

Custom cells

- New Snow Leopard delegate API to customize them

```
- (void)browser:(NSBrowser *)browser setObjectValue:(id)object
                                             forItem:(id)item;
- (CGFloat)browser:(NSBrowser *)browser heightOfRow:(NSInteger)row
                                          inColumn:(NSInteger)columnIndex;
```

Item-Based NSBrowser

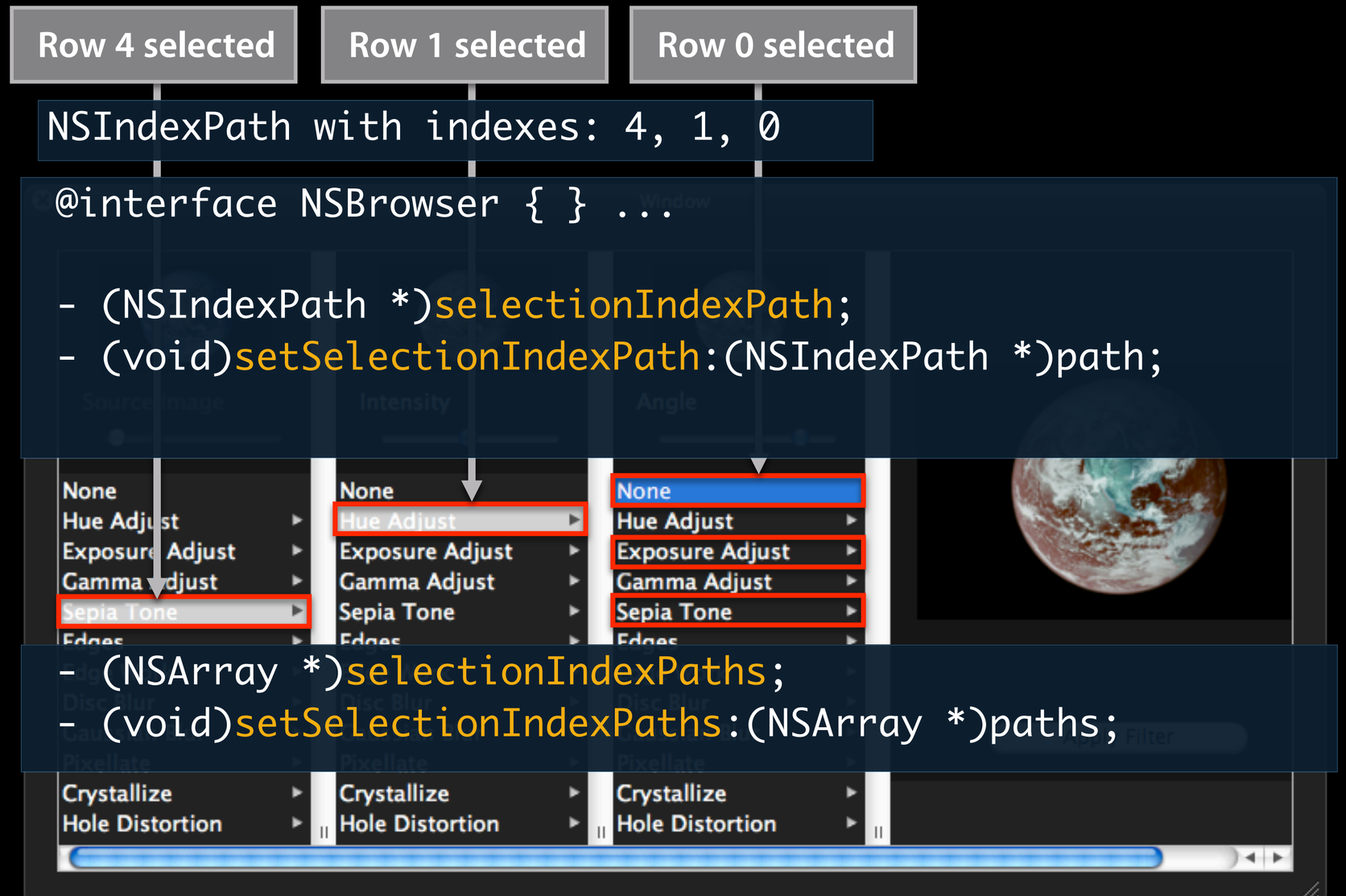
Custom cells

- New Snow Leopard delegate API to customize them

```
- (void)browser:(NSBrowser *)browser setObjectValue:(id)object
                                             forItem:(id)item;
- (CGFloat)browser:(NSBrowser *)browser heightOfRow:(NSInteger)row
                                          inColumn:(NSInteger)columnIndex;
```

Item-Based NSBrowser

Useful methods—Identifying a selection path

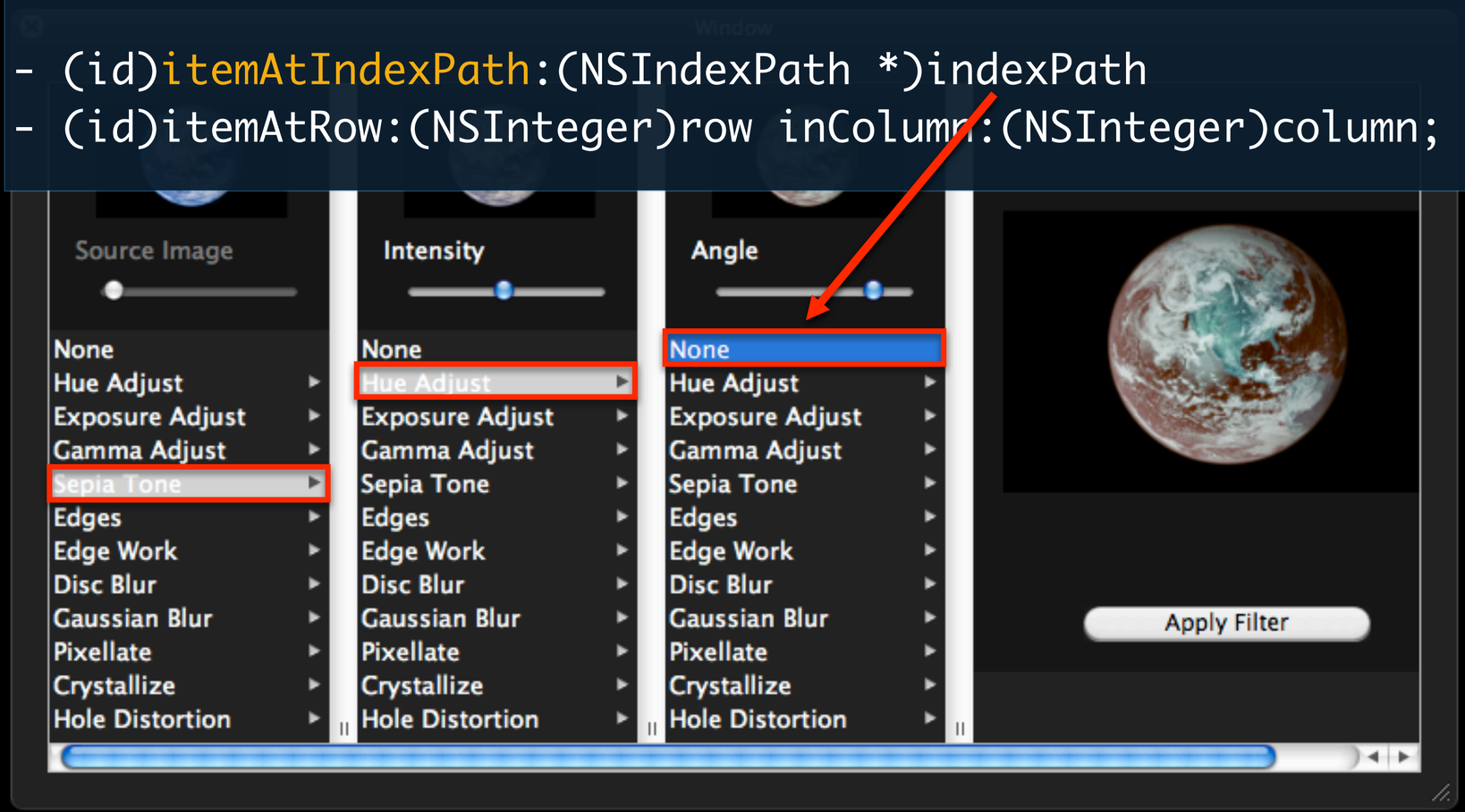


Item-Based NSBrowser

Useful methods—Identifying items at a path

```
NSIndexPath with indexes: 4, 1, 0
```

```
@interface NSBrowser { } ...

- (id)itemAtIndexPath:(NSIndexPath *)indexPath
- (id)itemAtRow:(NSInteger)row inColumn:(NSInteger)column;
```

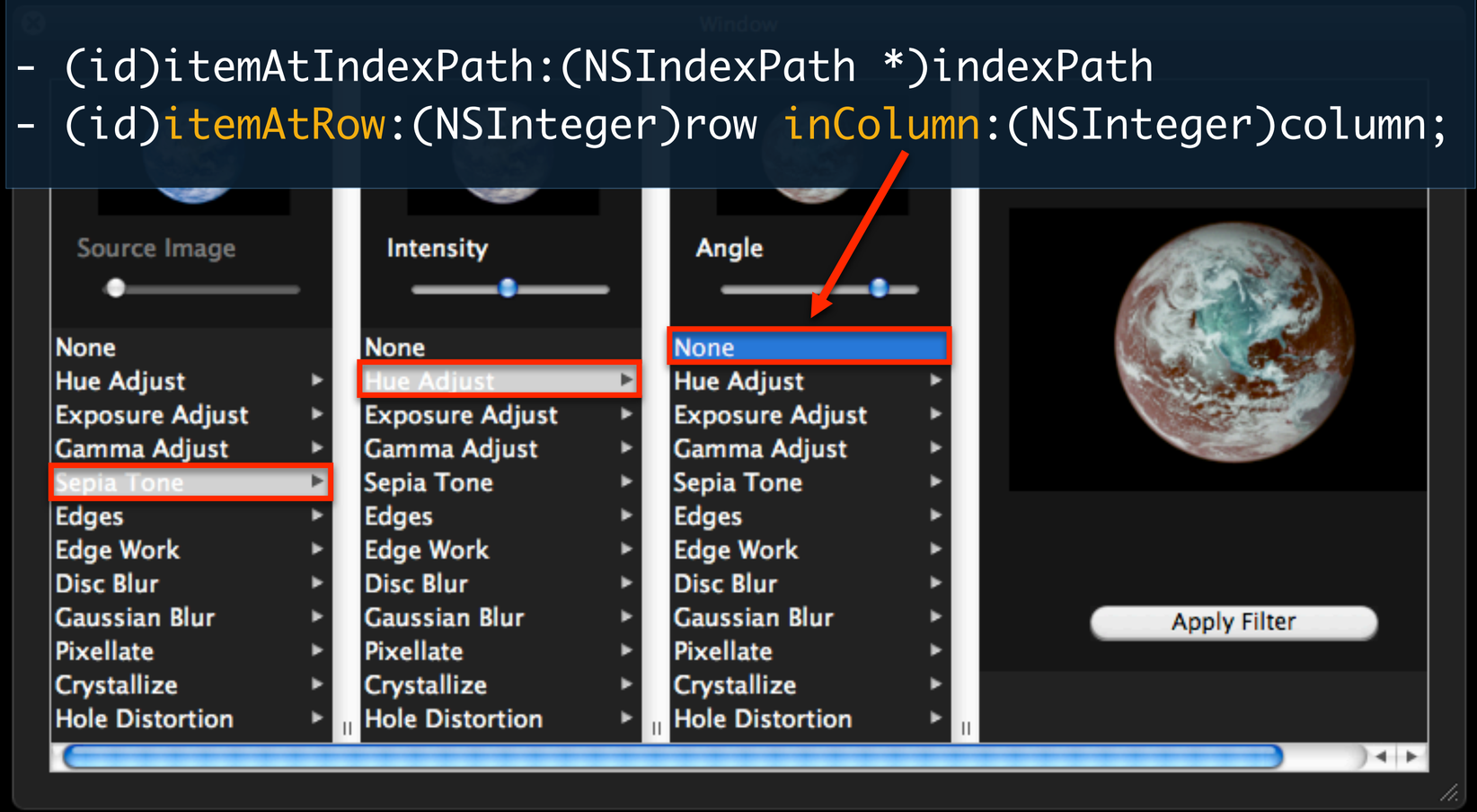


Item-Based NSBrowser

Useful methods—Identifying items at a path

```
NSIndexPath with indexes: 4, 1, 0
```

```
@interface NSBrowser { } ...

- (id)itemAtIndexPath:(NSIndexPath *)indexPath
- (id)itemAtRow:(NSInteger)row inColumn:(NSInteger)column;
```



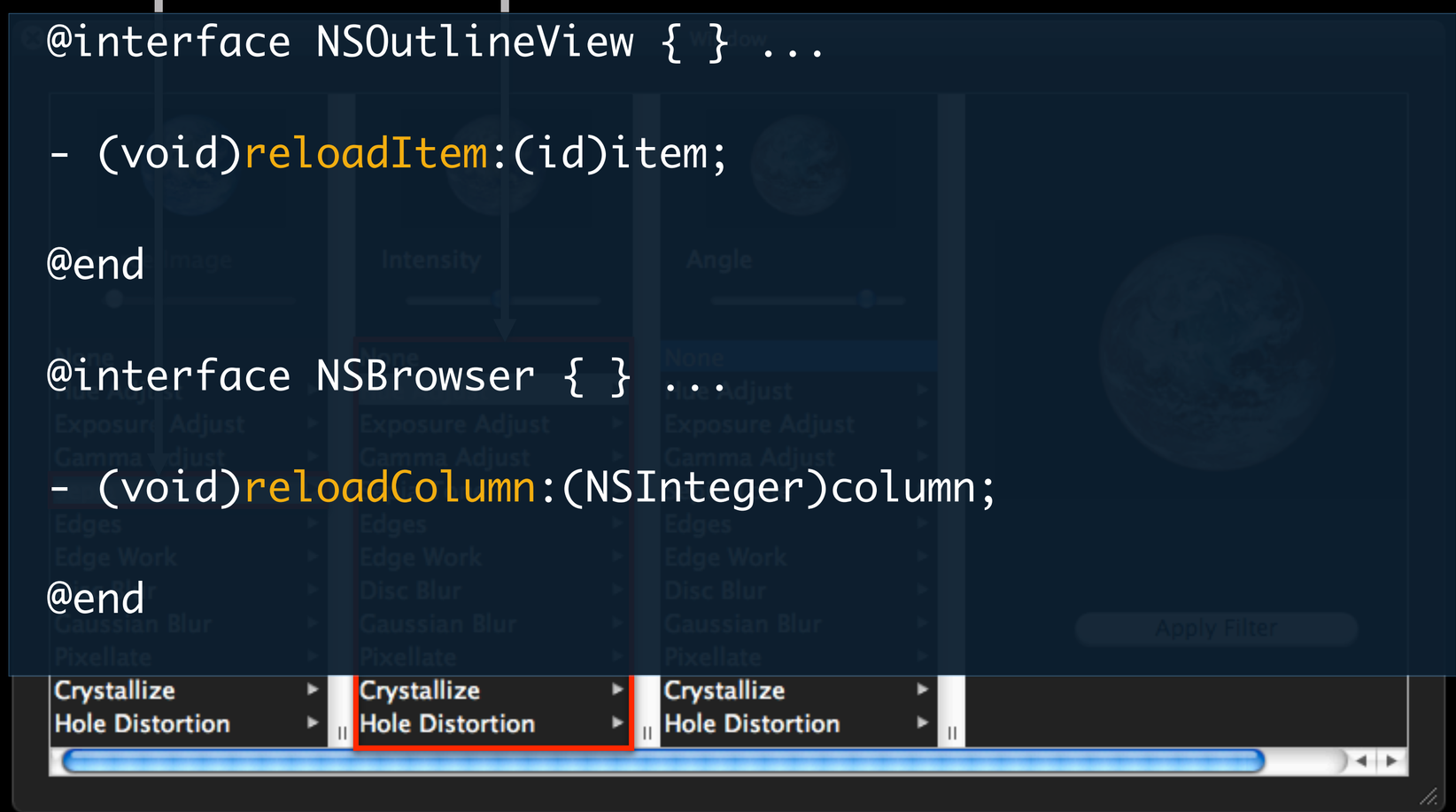
Item-Based NSBrowser

Useful methods—Reloading all children

Parent Item

Children in column 1

```
@interface NSOutlineView { } ...

- (void)reloadItem:(id)item;

@end

@interface NSBrowser { } ...

- (void)reloadColumn:(NSInteger)column;

@end
```



Conclusion

What We Are Covering

- Subclassing and enhancing NSTableView
  - Batch loading cell contents
  - Adding subviews
  - Custom cell editors
- Custom animations
  - Animating from a table view
  - Animating a pop-up window
- Custom NSBrowsers
  - Utilizing new Snow Leopard APIs

What We Are Covering

- Subclassing and enhancing NSTableView
  - Batch loading cell contents
  - Adding subviews
  - Custom cell editors
- Custom animations
  - Animating from a table view
  - Animating a pop-up window
- Custom NSBrowsers
  - Utilizing new Snow Leopard APIs

Conclusion

Use the sample code!

- “AnimatedTableView”—Available at http://developer.apple.com
- Modify and reuse the code

Labs and Related Talks

Cocoa Open Lab	OS X Lab C Thursday 2:00M
Cocoa Tips and Tricks	Nob Hill Thursday 5:00PM

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09